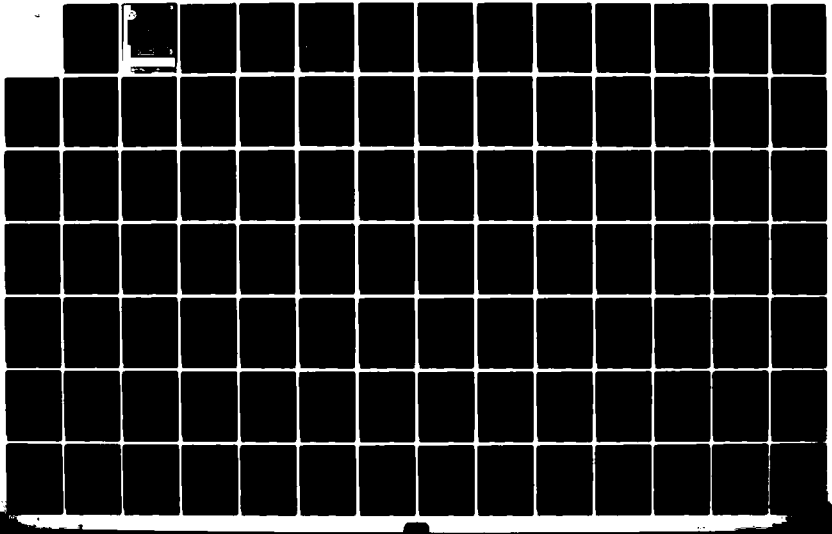
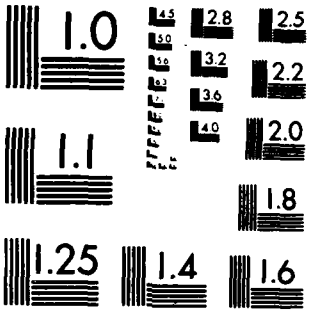
AD-A139 019 ALTERNATIVE KNOWLEDGE ACQUISITION INTERFACE STRUCTURES 1/2
(U) PERCEPTRONICS INC MENLO PARK CA KNOWLEDGE SYSTEMS
BRANCH K T WESCOURT ET AL. DEC 83 PPAFTR-1131-83-1
UNCLASSIFIED NAVTRAEQUIPC-82-C-0151-1 F/G 5/9 NL





[redacted]

[redacted]

Keith T. Vancourt and Perry W. Thorndyke
Perceptronics, Inc.
Advanced Systems Branch
545 Middlefield Road, Suite 140
Menlo Park, CA 94025

[redacted]

[redacted] 1983

Final [redacted] Report
[redacted] 1982 to September 1983

GOVERNMENT RIGHTS IN DATA STATEMENT

Reproduction of this publication in whole or in part is permitted for any purpose of the United States Government.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NAVTRAEQUIPCEN 82-C-0151-1	2. GOVT ACCESSION NO.: AD-A139019
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (and Subtitle): ALTERNATIVE KNOWLEDGE ACQUISITION INTERFACE STRUCTURES	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report Sep 1982 to Sep 1983
	6. PERFORMING ORG. REPORT NUMBER: PPAFTR-1131-83-1
7. AUTHOR(s): Keith T. Wescourt, Perry W. Thorndyke	8. CONTRACT OR GRANT NUMBER(s): N61339-82-C-0151-1
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Perceptronics, Inc., Knowledge Systems Branch, 545 Middlefield Road, Suite 140, Menlo Park, CA 94025	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Training Equipment Center, Orlando, FL 32813	12. REPORT DATE: December 1983
	13. NUMBER OF PAGES: 99
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): DCASMA Van Nuys, 6230 Van Nuys Blvd., Van Nuys, CA 91408	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Knowledge Acquisition
Artificial Intelligence
Training & Simulation
Expert Systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research developed a design concept for an interactive system to acquire domain knowledge from a training expert. Such a system would facilitate the development of knowledge-based instructional systems by directly eliciting and encoding from domain experts knowledge needed to deliver instruction. An analysis of the process by which knowledge-based systems are constructed indicates that (1) the generality of a knowledge acquisition system must be limited by domain characteristics and by the architecture of the system it

20. ABSTRACT CONTINUED

serves, and (2) the non-sequential, interacting activities during system development constrain the potential role of automated knowledge acquisition aids. A feasible concept for knowledge acquisition technology, building on prior research in artificial intelligence, involves the notion of class-generic systems for a related set of domains with a fixed architecture and training capabilities. This concept is developed and discussed in the context of proposed Navy training systems for acquiring (1) models of trainee performance during learning, (2) rules of behavior for an automated opponent in a tactics trainer, and (3) a knowledge base of facts to be subsequently presented to trainees for memorization. Data obtained from Navy domain experts and system builders indicate that the utility of knowledge acquisition systems to Navy domain experts will depend primarily on user skills and motivation and on the amount of conceptual support provided by the system's user interface. In contrast, low-level details of interaction medium and protocol are anticipated to be of secondary importance.

SUMMARY

Many Navy tasks depend on the skilled performance of highly-trained individuals. Training for such tasks often requires closely supervised, intensive, and highly technical instruction from a knowledgeable instructor. In many cases, skilled Navy instructors are in short supply--and the quality of instruction can vary from individual to individual.

Recent advances in computer science and psychology have created the opportunity to develop "intelligent" computer-based instructional systems. Such systems can not only identify trainee errors during instructional sessions but also understand why the trainee has erred based on its knowledge of the task and a model of the trainee's cognitive processes. Such diagnoses are critical to correcting a trainee's misunderstanding or specific skill deficiency about the task he is trying to learn.

The development of such computer-based instructional systems requires the accumulation and codification of extensive knowledge about the target task and trainees' behavior when learning the task. Such knowledge--essentially that which an expert instructor brings to bear during training--includes the skills and procedures required to perform the task correctly, the types of errors trainees can exhibit during learning, techniques for diagnosing trainee performance deficits, and instructional interventions to correct skill deficits and train appropriate behavior.

Development efforts aimed at constructing such knowledge-intensive "expert" systems often adopt the "knowledge engineering" paradigm. Knowledge engineering is an iterative process with six principal phases. These phases include definition of the system's capabilities, extraction of domain knowledge, formalization of the knowledge, design of the expert system, implementation of the system, and test of system performance. The process is distinguished by the necessity of repeated and extensive interactions between a domain expert and a highly trained system developer who elicits and encodes the expert's knowledge about task performance and training procedures.

The shortage of experienced knowledge engineers and the difficulty of co-locating knowledge engineers and domain experts for extended periods of time limit the opportunities for the development of new expert training systems. Therefore, a need exists to develop automated tools for knowledge acquisition and formalization that could interact directly with domain experts and reduce the involvement of knowledge engineers. Such tools could facilitate the more rapid and widespread development of advanced training systems with automated instructional features. These systems, in turn, would contribute to Navy training effectiveness by increasing the availability of high-quality instruction through the use of computer-based training aids.

The generality of current tools for automated knowledge acquisition is limited both by the present state of the art in knowledge engineering and by dependencies among stages in the process that make it difficult to isolate individual functions. The process of acquiring domain-specific expert knowledge is dependent on the formalisms used to represent that knowledge, which, in turn, depend on how the knowledge will be used. The technology for engineering knowledge-based systems has not yet progressed to a stage in which

relationships among domain characteristics, system objectives, and knowledge representation requirements are well understood. This lack of understanding limits the generality of system architectures and of any potential aids for the system building process. To a large extent, the power of such aids already developed has been inversely related to their generality.

The design of automated aids is also complicated by the opportunistic, unpredictable nature of the system development process. The opportunistic nature of the process derives from the multiple dependencies among the various stages in system development. The system builder has a number of interdependent objectives in addition to domain knowledge acquisition, and he must pursue them iteratively and incrementally--that is, in parallel--since there is inadequate design knowledge for achieving them sequentially. It is difficult to design an automated aid to be used under such variable, unpredictable circumstances. These characteristics of the knowledge engineering process limit the scope and generality of a knowledge acquisition system that can be produced given the current state of the art.

Within these constraints, project research considered a number of alternative concepts for aids to assist the development of knowledge-based instructional systems. These concepts were evaluated against a number of criteria that considered their feasibility, utility, and appropriateness for use in military applications. The most promising concept entailed the development of systems for the acquisition of knowledge in a variety of domains with a single conceptual class. The notion of a class is somewhat intuitive, but it may be defined more technically as a set of domains that share a significant number of concept abstractions among their bodies of knowledge and that could be taught by a single training system with a fixed set of instructional features. Such classes within the Navy include sonar and radar system operations, system maintenance, and platform-level combat tactics.

The development of an automated knowledge acquisition aid generic to a class of domains requires the prior development of an adequate training system for one domain in the class. Subsequently, the automated aid is developed and used to implement the "same" training system for other domains in the class. The automated aid depends on the particular implementation of the training system and on knowledge it embodies, abstracted from the first development effort, about how to elicit and organize domain knowledge for the domain class. This approach avoids the problem of interfacing an aid to an iterative and variable set of activities by completing all the activities, except the acquisition and encoding of specific domain knowledge, in the course of building the system for the first domain. The finished results of those activities are transported intact to the systems for the other domains. The approach follows from and builds upon the techniques used in the "skeletal systems" developed to assist implementation of expert consultation systems for specific classes of problems.

This report describes three alternatives for implementation of a class-generic knowledge acquisition system for use in developing advanced training systems. The first would implement an architecture and knowledge acquisition aid for eliciting from an expert the knowledge needed to perform automated performance diagnosis of a trainee during learning. The automated aid would acquire an instructor's knowledge of potential errors in task performance and

other performance deviations needed to support automated performance diagnosis during training sessions.

The second alternative knowledge acquisition aid would capitalize on ongoing efforts to develop "intelligent" automated opponents in tactics training systems. In this case, the aid elicits elaborations of alternative (possibly sub-optimal) opponent tactics and knowledge of when during training to invoke these alternative behaviors to achieve pedagogical objectives.

The third alternative extends a prior Navy research and development effort that implemented a prototype generic instructional system. That system uses instructional "games" to assist trainees in the memorization of domain facts and relations. It has already been applied to several different domains. However, building new domain knowledge bases for the system remains a costly and lengthy manual process. This alternative would aim therefore at implementing a more cost-effective automated approach to building knowledge bases for this existing prototype system.

These three alternatives for pursuing the concept of a class-generic automated knowledge acquisition aid represent a range of tradeoffs among issues involving cost, payoff, and feasibility. The tradeoffs cannot be resolved on purely technical grounds in favor of any one of the alternatives. Instead, Navy priorities will need to be considered in selecting an alternative for further development.

Analysis of the knowledge engineering process and human factors considerations led to a set of guidelines for the development of user interfaces to these systems. These guidelines emphasize system characteristics required to insure the utility of the knowledge acquisition aid to Navy domain experts.

Discussions of user interface design issues for the class-generic knowledge acquisition aid were held with several Navy domain experts and training system developers. These discussions provide some guidance for the pursuit of any of the alternatives. The most significant and consistent opinion expressed indicated that the medium of human-machine interaction--frequently the focus of so-called "user-friendly" interface design--is not likely to be the crucial factor in determining the utility of a knowledge acquisition system to Navy domain experts. Rather, discussants emphasized the need for conceptual support for the knowledge specification task in the user interface and the importance of selecting potential users with some computer skills and high motivation to use the system. They perceived high motivation to contribute to a system development effort as the sine qua non for effective automated knowledge acquisition: the best user interface would not be sufficient to support use by domain experts arbitrarily assigned to work with a system development team. Given high motivation and some computer skills (which are increasingly widespread among Navy personnel), conceptual support from the user interface becomes the most critical design issue.

An interface to a knowledge acquisition system should provide several types of conceptual support. These include (1) adaptive control of dialogue initiative, (2) user access to information about prior and potential future contexts for his activities with the system, and (3) feedback about how knowledge supplied by the user affects the behavior of the training system.

The architecture of the knowledge acquisition system and its interface should therefore include intelligent dialogue planning and interaction management functions. Further, it should provide a flexible interface to the target training system for experimentation with the incrementally developed knowledge base supplied by the user. Detailed design decisions regarding interaction media and protocols can only be resolved when the characteristics of the domain class and features of the training system's own architecture have been identified.

Two activities therefore emerge as critical steps toward the development and implementation of any of the three alternative knowledge acquisition concepts. First, research must focus on the problem of characterizing and representing class-generic knowledge necessary to capture training expertise in a variety of domains. Second, techniques must be devised to use this knowledge in an appropriate system architecture providing conceptual support for the user during knowledge acquisition sessions. Further design of low-level interface details can be deferred at least until a prototype system accessible to motivated, skilled users has been developed.

PREFACE

The application of knowledge-based modeling techniques to training simulators appears likely because of increasing pressures to make these devices more "intelligent." There is an emerging requirement to decrease the instructor/student ratio for simulator-based training. This requirement is dictated by logistical considerations. In order to counter this potential threat to training effectiveness, intelligent training devices implemented through knowledge-based models could be developed to augment the instructor cadre. Instructor functions would be provided in the form of software models. This will allow the instructor/student ratio to be reduced without reducing the amount or quality of instructional guidance.

The Human Factors Laboratory at the Naval Training Equipment Center has been pursuing research related to the development of intelligent training devices for over a decade through research programs in adaptive training, student performance measurement, and part-task training. It is becoming apparent that such functions can most effectively be implemented using concepts borrowed from the artificial intelligence research community. Attention is being focused on knowledge-based models, in particular, expert systems.

The current development cycle of an expert system is very lengthy and consumes a great deal of resources. Making the implementation process more efficient would support the application of these knowledge-based models to simulator-based training. In particular, the knowledge engineering process has to be streamlined. The present task was initiated to investigate the possibilities for reducing resource expenditures during the process of knowledge engineering. The stated goals were to determine the extent to which the process could be automated and to make recommendations concerning the conditions under which such automation would be practical. Clearly, knowledge engineering is a complex cognitive activity and a general, completely automated procedure cannot be supported by the current state of technology. However, it appears that a workable system can be developed to automate certain phases of the knowledge engineering cycle for particular classes of expert models. This report details the procedures followed in reaching this conclusion.

Robert Ahlers
Scientific Officer

ACKNOWLEDGEMENT

We are grateful to the personnel of the Fleet Combat Training Center, Pacific and the Navy Personnel R&D Center for their participation in the discussions summarized in Section V. We also wish to acknowledge the role of the colleagues who provided informal feedback on our analysis of the knowledge engineering task and concept for introducing automated aids for it.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

BACKGROUND

Much training in the Navy and other military services requires closely supervised, intensive, and highly technical instruction on a complex task. In many cases, such instruction is provided by a training specialist in conjunction with a training simulator. Simulators are typically designed to develop and extend knowledge and skills that are impractical, expensive, or impossible to exercise within operational environments. Typically, such simulators provide practice on the target task but little or no instructional feedback on trainee performance, skill deficiencies, or coaching on correct behavior. Human instructors must therefore observe trainee performance and provide appropriate instructional interventions. However, in many cases, skilled Navy instructors are in short supply relative to the number of trainees, and the quality of instruction can vary from individual to individual.

Recent work on "intelligent" simulators is leading to the development of simulators with instructional capabilities in addition to simulation of operational equipment and situations. These training simulators will embody "surrogate instructors", which, in conjunction with human instructors, could better provide trainee performance evaluation and adaptive training. Such augmented training capabilities can significantly increase the cost-effectiveness of simulator-based training and extend the availability of individualized instruction.

Research on surrogate instructor technology (also called Intelligent Computer-Assisted Instruction [ICAI]) has utilized artificial intelligence techniques to represent conceptual knowledge about the problem domain, expert and trainee performance, and instructional methods. Many of these techniques are derived from those used in so-called "expert systems"--knowledge-intensive, high-performance programs designed to serve as automated consultants to domain experts. A recognized bottleneck in the development of expert systems and hence surrogate instructor systems is the human resources, time, and cost required to articulate the expert domain knowledge and to encode it in software. Current approaches involve frequent, long-term interactions among a team of highly-trained knowledge engineers and domain experts. While some technology has been developed to assist knowledge engineers in developing expert consultation systems, it has not reduced requirements for person-to-person interactions that account for much of the development time and cost. The application of technology assistance to developing surrogate instructor systems has lagged behind expert systems development, and little work has addressed the requirements for surrogate instructional systems insofar as they differ from expert consultation systems. Therefore, a need exists to identify technological opportunities to facilitate and streamline the task of articulating expert knowledge to be used in the development of an automated instructional system.

APPROACH

The current project sought to define a set of feasible, high-payoff, research objectives for the automation of the knowledge elicitation process. As a first step, a review was made of the available documentation of previous and current expert and instructional system building efforts. Particular attention was given to those projects attempting to develop tools to aid the system building process and to provide generic system capabilities. Conversations with other knowledge engineers provided insights into the difficulties and pitfalls of designing expert systems. In addition, these individuals provided useful comments on the design concepts developed to meet NAVTRAEQUIPCEN requirements. Finally, interviews with domain experts and training specialists in the Navy elucidated system design constraints dictated by characteristics of target users and the operational environment.

Early on in the review and analysis effort, it became apparent that the present state-of-the-art in the field of expert systems can not support a detailed generic design for surrogate instructor systems. Variations across domains and desired system capabilities require different representations for knowledge and different mechanisms for applying it. As of now, no single, uniform representation of concepts, relations, procedures, and strategies has been found sufficient to capture domain expertise in a wide variety of domains.

In addition, it became clear that the software modules in a surrogate instructor system that use the domain knowledge can not be independent of the representations or their use. This dependence also extends to software that would aid the development of surrogate instructor systems. In particular, details of an effective user-interface design for acquiring domain knowledge from an expert depend on the knowledge representation. Generic human engineering principles for interface design are only rough guidelines to system development, and their application requires more detailed interpretation with respect to the target system's specific features and implementation. Thus, we concluded that an intelligent, generic system intended to support the development of any surrogate instructor system was not feasible given the current technology and state of knowledge in expert systems.

We therefore worked to determine a more restricted concept of generality for an automated knowledge acquisition system. Using this concept, we attempted to elucidate (1) a design for a knowledge acquisition system specified to the extent possible without committing to a particular surrogate instructor system architecture, (2) a set of alternative interface concepts consistent with the design for which additional work could be realistically undertaken to produce a useful system, and (3) a set of issues that must be resolved in such additional work.

The concept we developed addresses generality for an automated knowledge acquisition system at the level of a class of tasks, each member of which can be adequately served by a fixed set of surrogate instructor capabilities and knowledge representation formalisms. The notion of such classes is largely intuitive: a characteristic of members within a class is congruence of high-level semantic and pragmatic aspects of domain knowledge, which we will refer to as class-generic knowledge. However, regardless of its intuitive nature,

in our work we have found that consensus exists for the definition of some such classes of tasks.

This report presents the results of our research to specify and elaborate a design for an instructional knowledge acquisition system. Section II presents a model of the knowledge engineering process and describes the activities required to build an intelligent instructional system. This section indicates how particular features of that process preclude the full generality NAVTRAEQUIPCEN sought in its original concept. Section III reviews selected research on tools to assist expert system building. This review illustrates more concretely the factors that limit generality. Section IV presents our general system concept, and a set of criteria to be considered in evaluating alternative realizations of the concept. Three alternative specific concepts are then presented that differ with respect to these criteria and to the specific instructional capabilities of the systems they serve. Section V introduces a set of user interface design issues and describes interviews conducted with system builders and domain experts to determine how those design issues might be resolved in realizing our system concept. Section VI presents an architecture for implementing those functions and interface features of the system concept that can be specified without further commitment to the class of tasks and the host system's capabilities. Finally, Section VII reviews the conclusions and recommendations derived from the research.

SECTION II

THE KNOWLEDGE ENGINEERING TASK

Planning effective automation for aspects of any complex task requires an analysis of the task and development of a process model of the task. This model is required both for determining what task components to automate and how that automated system should interface to the people who will use it. This section presents an analysis of the knowledge engineering task and a model that describes the relationships among task objectives, the activities that attain them, and the knowledge required by those activities.

WHAT IS KNOWLEDGE ENGINEERING?

Knowledge engineering is the process by which a class of computer programs called expert systems are created. These systems are built to aid or perform tasks that are very knowledge intensive, typically require inexact and imprecise reasoning, and for which the expertise for performing the task resides primarily with a very few human "experts." These tasks involve some form of situation interpretation, decisionmaking, and/or planning. The best-known examples of recent expert systems include programs for medical diagnosis (Shortliffe, 1976), chemical analysis (Lindsay, et al., 1980), geological analysis (Duda, et al., 1978), and planning computer system configurations (McDermott, 1980). Knowledge engineering differs from conventional software engineering in (1) the nature and extent of the interaction between the knowledge engineer(s) (KE) and the domain expert(s) (DE), and (2) the types of software design and implementation tools used.

The KE is more dependent on the DE both prior to implementation and for post-implementation testing than is typical in other software engineering efforts. Because expert systems are knowledge intensive, knowledge elicitation is one of the KE's major technical objectives. It alone generates a need for frequent and prolonged interaction between the KE and DE prior to and during implementation.

Knowledge acquisition has proved to be a complex and difficult process. The knowledge that must be incorporated into an expert system is largely non-numerical and imprecise. It is usually expressed as an extensive body of concepts, rules, and approximate methods. The term heuristic is used to describe both this type of expert knowledge and the type of programming a KE uses to operationalize it. The expert's heuristic knowledge is often tacit and thus difficult both to elicit and articulate. Determining whether the evolving knowledge base is consistent and when it is complete enough is therefore a major problem for the KE and DE. This fact and the imprecision of heuristic knowledge increase the KE's dependence on the DE for debugging, evaluating, and tuning performance of an expert system. Thus, knowledge acquisition typically continues well into the implementation stage.

The data types and control structures of conventional programming formalisms are not conceptually well-suited for describing heuristic knowledge and for developing expert systems. They do not provide an organizational framework for knowledge acquisition and lack good facilities for debugging

knowledge-intensive code. Research on expert systems has therefore evolved representation languages, symbolic programming languages, and rule-based problem-solving architectures to allow more intuitive and transparent operationalization of heuristic knowledge on computers. In addition, specialized system-building tools have been developed (e.g., Davis [1977], Reboh [1981]) to aid the KE in encoding knowledge into specific representations and in testing and refining an implementation. (Section III reviews these design and implementation tools.) These tools are designed to be used interactively; hence, they enable and support the incremental system building and testing necessary in expert systems.

THE PRACTICE OF KNOWLEDGE ENGINEERING

As a research area, knowledge engineering--a sub-field of artificial intelligence (AI)--is about 15 years old. The total number of practitioners is under 300 internationally, mostly located in universities. Only very recently have serious applied and commercial development efforts outside academic research centers been undertaken. The academic focus has been on the individualistic, innovative, and high-risk features of research in an emerging field rather than on systematically reducing to practice the process of knowledge engineering. As a result, the construction of expert systems is still more of an art or craft than a discipline.[1]

Knowledge engineering is a highly intellectual and individualistic process, although it is not uncommon for several KEs to work together on a project. KEs generally hold graduate degrees either in computer science or cognitive psychology and are familiar with the concepts and methods of AI. However, they receive little formal training on how to build expert systems. Instead, they acquire the necessary skills, usually as graduate students, in a loose apprenticeship system under the supervision of more experienced practitioners. As in any apprenticeship program there is considerable variability in supervision. Progress in learning knowledge engineering through hands-on experience is impeded by a lack of precise criteria for assessing a KE's performance. Furthermore, it is not obvious at what point someone becomes a qualified knowledge engineer.

Without exception, documentation of existing knowledge engineering efforts describes numerous false starts and revision cycles prior to achieving a system of any practical value. This characteristic may reflect in part the academic research setting of the efforts and its tendency to encourage inventiveness and discovery of alternatives even after existing efforts have demonstrated workable methods and tools. However, there are substantive limitations on the process of building expert systems that are more important for explaining its characteristics. First, there are problems in formulating initial system specifications; existing experience is either too limited or

---

[1]Arguably, the same remark could be made about other software engineering as well. However, this field is more mature and considerable effort during the past decade has attempted to organize and standardize the production of commercial software.

has not been sufficiently analyzed to determine exactly what an effective expert system for a task domain should do. Second, there are problems in implementing system specifications; there is a similar lack of experience or understanding regarding the relationship between a set of specifications and appropriate methods and tools for achieving it. At least at this point in time, the knowledge available to the KE from prior efforts for generating and evaluating designs is heuristic--as heuristic as the domain knowledge he himself must acquire and operationalize to build an expert system. This use of heuristic knowledge by the KE accounts in part for the iterative nature of the knowledge engineering process. It also suggests that analyzing and modeling what the KE does, in order to consider how to automate aspects of knowledge engineering, may itself be viewed as an exercise in knowledge engineering. However, the scope of the knowledge engineering task is considerably broader than that of any task for which an expert system has yet been built. Modeling the knowledge engineering task is therefore important to permit identification of smaller, relatively independent activities that might be feasible candidates for automation.

A COGNITIVE TASK MODEL OF KNOWLEDGE ENGINEERING

The design of automated knowledge engineering functions first requires a cognitive task model specifying the task's activities and the relationships among them. The naive approach to introducing automation suggests that where implementation is technically feasible and cost-effective, the machine should perform all activities in which its productivity exceeds that of the human. This approach overlooks the need to consider the nature of the human-machine interaction that must occur. When activities are divided, those that are retained by the human may include some that depend on knowledge and information generated by activities assigned to the machine. Likewise, the machine may be dependent on knowledge generated from the human's activities. A cognitive task model specifies how tasks depend on knowledge and information generated or modified by other tasks. Thus, the model can be used to determine what knowledge must pass the interface between human and machine and the frequency with which that interface must be used. It can show that although some tasks can be performed better in isolation by the machine, the cost-effectiveness or viability of the overall system requires that those activities be retained by the human. Although human performance is remarkably flexible, ill-considered human-machine architectures can overwhelm human cognitive capacity and endurance or impair motivation for using the system.

A cognitive task model is therefore an important tool in determining both what task components to automate and how to design the human-machine interface. It enables an understanding of how the attainment of objectives is shaped by requirements for knowledge, performance factors and external constraints. This understanding permits a design for automation that can enable a human-machine "team" to to attain the task objectives more effectively or efficiently than can a human working alone.[2] The model of the

[2]It might also lead to a conclusion for some tasks that automation would not be feasible or cost-effective.

knowledge engineering task postulates activities performed by three participants: the customer (the ultimate end user and perhaps the financial sponsor of the knowledge engineering product), the knowledge engineer (the designer and implementor of the system), and the domain expert (the source of domain-specific knowledge).

Figure 1 summarizes the knowledge engineering process as the interacting objectives and activities of these participants. It presents the set of tasks as a flow chart indicating the ordering of and dependencies among the various activities involved in a knowledge engineering effort. Because we have summarized the process in flow chart form, the objectives and tasks may appear to have a "natural" linear order. However, this appearance is deceiving, for it presupposes a mature design science for building expert systems. The ability to design a system successfully (i.e., so that the first implementation of the design performs acceptably) requires (1) a set of general design principles, (2) knowledge required to apply the design principles to the particular problem at hand, and (3) a method to determine that the resulting design is complete and satisfactory. In the knowledge engineering process, there is no comprehensive or generally accepted body of knowledge for meeting any of these requirements. Therefore, achieving a workable design most typically requires the interaction among component processes of design, implementation, test, refinement, and redesign. No one of these processes proceeds in isolation; rather, several are simultaneously active and under consideration. Thus, a linear stage model has inherent limitations as a description of this complex process.

An alternative to the linear model is a class of models that accommodates the simultaneous operation of multiple cooperating processes. These models, called blackboard models, have been used to model other complex cognitive processes such as planning (Hayes-Roth, 1980; Thorndyke, McArthur, and Cammarata, 1981), decision making (Thorndyke, 1982), speech understanding (Erman, et al., 1980), reading (Rumelhart, 1976), and sensor interpretation (Nii, et al., 1982). We believe that an attempt to develop a detailed computational model of the knowledge engineering task might profitably adopt this modeling approach, since the non-deterministic order of activities is easily accommodated by this framework. However, one limitation of blackboard models is the difficulty of representing and illustrating succinctly the individual activities required for task performance or the relationships among them. In this respect, the linear model is superior in its ease of illustration.

Since a clear explication of the knowledge engineering process is fundamental to the subsequent understanding of recommended proposals for automation of components of that process, we have chosen to illustrate the knowledge engineering process as a linear model, as shown in Figure 1. However, we recognize that the nominal sequential order implied by the figure is not a strictly accurate characterization of the way in which knowledge engineering proceeds. In recognition of the non-determinism in the order of activities, we have used an unconventional notation in the flow chart shown in Figure 1. Several of the branches leading from decision boxes lead to multiple points. This notation indicates that any one of the indicated activities can be undertaken next depending on conditions not represented in the flow diagram.

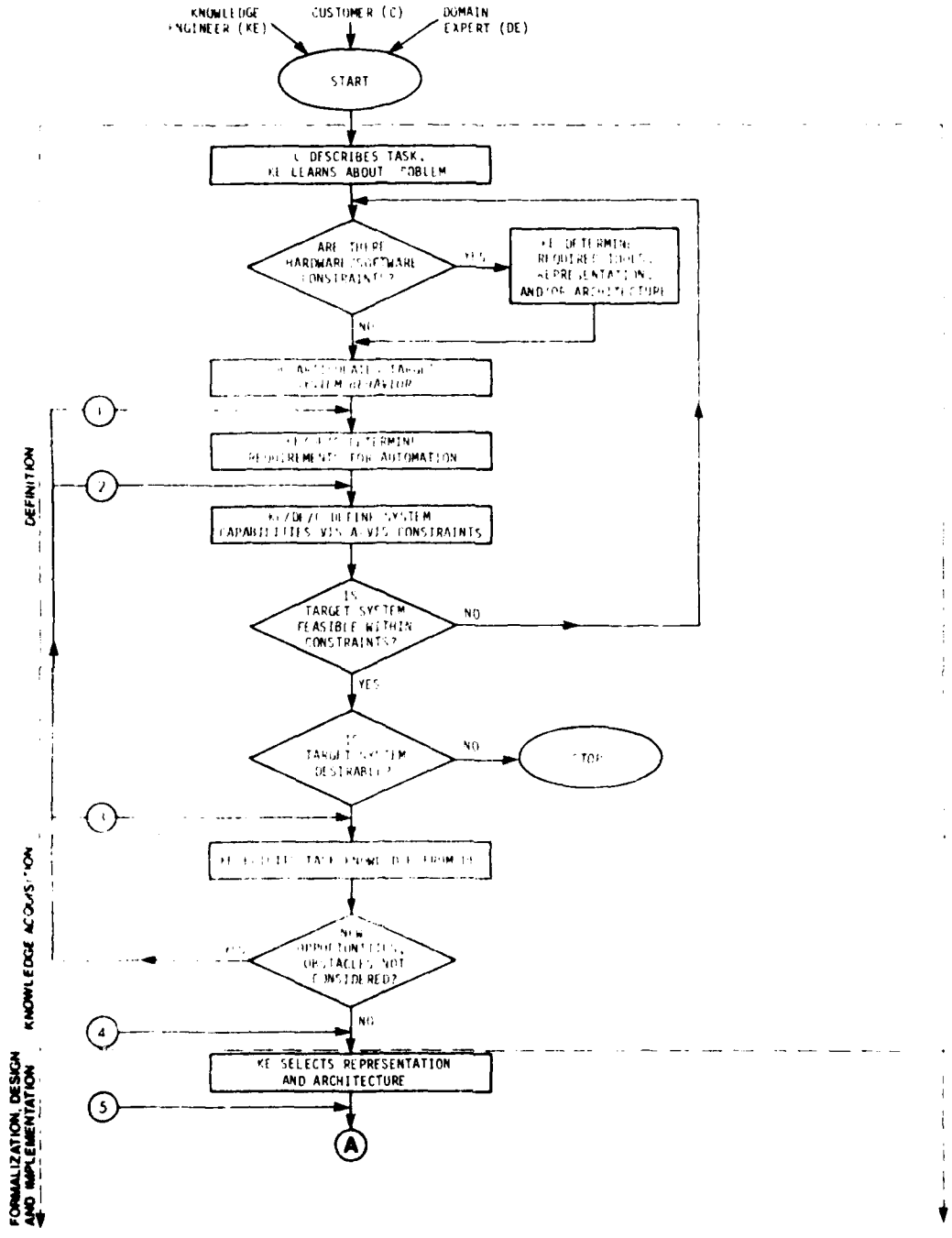


Figure 1. Control flow of knowledge engineering task model

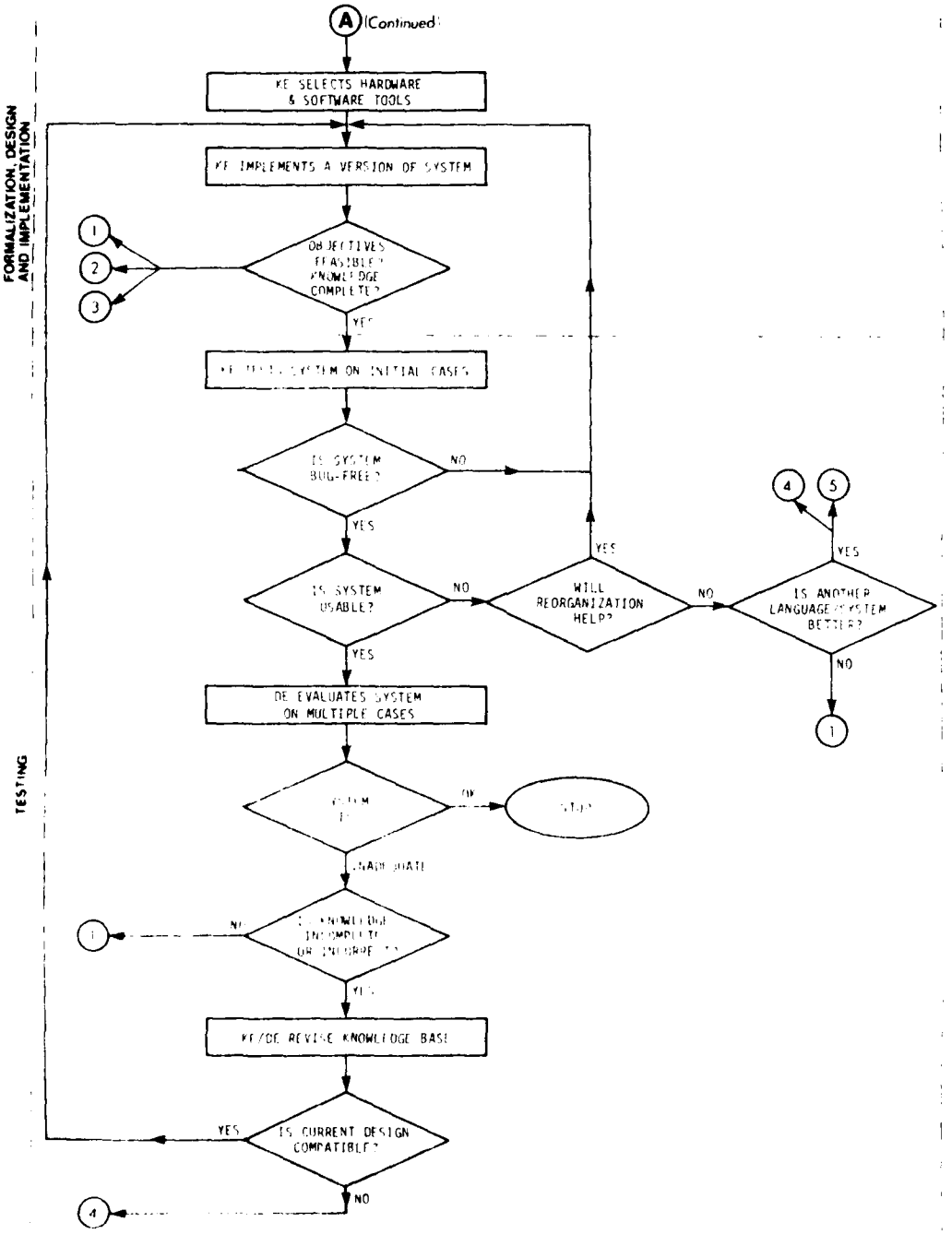


Figure 1 (cont.). Control flow of knowledge engineering task model

Strictly speaking, the KE's primary objective is to create an operational expert system that solves a problem for his customer. Six major technical objectives must be attained to create an operational expert system:

a. Definition: Specification of the capabilities the system will exhibit.

b. Knowledge Acquisition: Description of the knowledge--concepts, facts, and problem solving methods--needed to achieve the specified capabilities.

c. Formalization: Organization of the knowledge using formal representation languages.

d. Design: Determination of hardware and software architecture.

e. Implementation: Implementation of the encoded knowledge, procedures, and user interface.

f. Testing: Evaluation and refinement of the system.

These six objectives are similar to those described elsewhere (see, for example, Buchanan, et al. [1983] and Reboh [1981]). In examining how the six objectives have been addressed in prior systems, it is important to note that most previously developed expert systems are products of R&D environments. In these efforts, creating a working system was often secondary to performing successful science (e.g., the development of new knowledge representations, problem-solving methods or system-building tools). This difference in the primary motivation of R&D and of potential product-oriented applications is important because it leads to different subobjectives for the six major technical objectives and entails different external constraints on the KE. For example, in academic research the objective of Formalization may entail inventing a new representation. In an applications effort, on the other hand, external constraints may strongly discourage the KE from inventiveness in favor of selection from existing representation languages. In fact, as we will explain later, he may even be constrained a priori to use a particular representation language. Thus, in developing a task model based on prior case studies, we took into account how academic research motivations affected pursuit of the objective of building a working system. By so doing, we have oriented the model toward use in designing expert systems in product-oriented environments.

The remainder of this section discusses the model in detail. We structure our discussion around the KE's six technical objectives listed above. For each, we first present its immediate subobjectives and then consider in detail

- the knowledge the KE brings to the task for attaining that objective
- its dependencies on both other technical objectives and external constraints

- the types of activities that achieve it

- the criteria by which the KE monitors progress on it.

A. Definition Objective

Subobjectives

Generate potential system capabilities
    Identify customer and DE problems
    Identify opportunities for technological enhancement
    Identify user interface requirements and constraints

Select target capabilities
    Identify cost-effective candidates
    Identify technologically feasible candidates
    Determine cognitively feasible capabilities

The Definition objective entails determining the performance capabilities of the expert system and how it will interact with users. A planning or decisionmaking task, for example, might require information interpretation, option generation, option evaluation, and option selection activities. An expert system might assist the user with any of these, depending on the approach to cooperative man-machine problem-solving the customer desires. In addition, cost-effectiveness, feasibility, and other constraints on the expected use of the system will influence which particular capabilities are selected.

The major subobjectives of capabilities definition are (a) generation of candidate capabilities and (b) filtering those candidates to produce a target set satisfying the various criteria. The system's desired capabilities are initially motivated by problems in performing the task. Ordinarily, these are apparent and have motivated the customer's initial decision to consider an expert system. They may be documented in written materials but can be most clearly defined through interactions between the KE, the customer and DE.[3] Typically, the improvement sought in the expert system entails lowering the cost, increasing the speed or reliability, and/or improving the accuracy of task performance relative to the current methods.

For the KE to comprehend the difficulties the customer and DE perceive in their task--and to evaluate differences in the perceptions of the customer and the DE--he needs some knowledge of the task and how it is performed. If the KE is initially ignorant about the task domain, his initial interactions with the customer and DE must provide an introduction to the domain. This introduction explores the general objectives and methods of the task as it is currently performed. The KE develops the foundations of a task model using

[3]We distinguish the customer, or sponsor, of the effort from the DE for the sake of generality. In particular cases, the customer and the DE may be the same individual, or the DE may represent the interests of the customer to the KE.

his knowledge of modeling frameworks. As he adopts a general framework, he can take some initiative in seeking additional information he needs. Much of the domain knowledge the KE elicits in these initial interactions will not be encoded in the ultimate system.

Once the KE has a basic understanding of the task domain, he can guide the exploration of the problems the customer and DE perceive and characterize their performance and cost attributes. Initiative by the KE is important at this point to focus the interactions. Together, the KE, customer, and DE can then arrive at a set of performance capabilities that would remedy the identified problems.

The customer and DE must also provide information about the intended users of the system and the environment in which it will be used. This knowledge is needed by the KE to determine alternative user interface capabilities for the system. The prior experience of the users with interactive computing systems, their motivation and ability to use the system, and their technical knowledge of the domain--if it is different from the DE's--are all considerations in designing the user interface. For instance, interface capabilities that depend on typed input can be inappropriate if users cannot be motivated to learn or use typing skills in their work environment. This particular fact has been learned too late by designers of some commercial software systems oriented toward managers.

Another type of constraint that may influence system capabilities and interface design involves hardware or software requirements. The customer may have cost considerations, existing equipment, or organizational policies that dictate the use of a particular system. If so, the capabilities provided by this system may either restrict or present opportunities for the use of certain interface media, software packages, and programming languages.

Once he has an understanding of the domain and the constraints on system development, the KE can generate additional candidate capabilities for the system based on technological opportunity. By recognizing that the task shares features with others for which expert systems have been built, the KE may propose to include capabilities present in those systems. The decision to include such capabilities may involve considerations not apparent to the customer and the DE. For example, the decision may be based on the KE's knowledge that the capability is an easy enhancement--one which may be virtually "free" because the knowledge it requires will already be used to implement another capability. The KE's ability to realize technological opportunity in generating capabilities is dependent on his knowledge of prior expert systems and his ability to draw analogies from features of those systems to the current task.

The second objective in capabilities definition is filtering the candidate capabilities to produce a target set. The primary selection criteria are the cost-benefit expectation for a capability, the technological feasibility of a capability, and the overall operational coherence of the target set. Cost-benefit evaluation depends on (1) the resources the customer is willing to allocate for developing and operating an expert system; (2) the KE's ability to estimate the development effort and computational resources a capability will require; and (3) the beliefs that the KE, the customer, and the DE have about the value of enhancing existing task performance.

Technological feasibility becomes a difficult issue when there are no precedents for implementation of a capability or when there are initial constraints. The KE must estimate the risk involved in pursuing development of the capability and a joint decision must be made about whether the risk is acceptable.

In addition to evaluating capabilities individually, the coherence of the entire operational concept must be considered. A piecemeal system with loosely-coupled component capabilities may be inefficient and difficult to use. The set of capabilities must be evaluated with respect to overall implementation and support requirements and to the task model the KE develops. The model is used to consider how dependencies among candidate capabilities should determine the composition of the target set. For example, suppose among a set of candidates were included an automated deduction capability and a hierarchical explanation capability. Suppose that for either cost-benefit or feasibility reasons a decision was made to eliminate or at least substantially change the deduction mechanism. If the purpose or expected benefit of the explanation capability depends in some way on the deduction capability, then it too must be eliminated or modified.

Interface capabilities must be considered again at this point as well. The interfaces to different capabilities must be coherent with one another and with user requirements. To the extent that the cost-effectiveness or feasibility of specific technical capabilities are linked to these interface capabilities, the technical capabilities may themselves need to be reconsidered. For example, suppose there is an option generation capability in which the user can supply constraints and an interface supporting quasi-natural language. The KE may determine that the interface will be inappropriate for expressing constraints. One or both capabilities may then need to be revised or eliminated from the target set. In that case, other capabilities that depend on them may need to be reconsidered as well. Thus, while individual capabilities may be filtered for singular reasons, each addition or deletion from the target set may entail ramifications for other capabilities already included or excluded.

According to one source (Buchanan, et al., 1983), difficulties in initially defining system capabilities are among the major causes of the inefficient process of iterative development and revision in knowledge engineering. A major contributor to these difficulties is the "knowledge gap" between the KE, the customer, and the DE. This gap is inherent in the knowledge engineering task, since a defining feature of an expert system is its codification knowledge known only to a few specialists. Only some of the DE's extensive task knowledge may be needed to implement system capabilities, but almost all of it is relevant to selecting which capabilities to implement. The DE may have difficulty articulating this knowledge, since it is tacit and ad hoc. The KE is an expert in modeling and programming formalisms unfamiliar even to those who use computers in more routine applications such as data processing and data base management. The customer and the DE may themselves have varying perspectives on the task and the reasons for seeking to apply expert systems technology. The customer is typically a manager who may not have current technical knowledge but who may have a perspective on the organizational context surrounding the DE's task. Definition of capabilities requires the exchange and integration of these different types of knowledge, perspectives, and technical vocabularies.

The difficulty of rapidly bridging the knowledge gap is thus a significant factor mitigating against a successful one-step effort to define system objectives. However, even if a single person qualified as KE and as DE for a task, other factors would cause revisions of initially targeted capabilities. These factors, to be elaborated in subsequent discussion of other technical objectives, involve the incomplete and uncertain nature of current expertise in knowledge engineering. During the phases of Knowledge Acquisition, Design, Implementation, or Testing, initial estimates of cost or feasibility may have to be modified, creating a need to modify the targeted system capabilities. Since Knowledge Acquisition, Formalization, Design, and Implementation depend on the targeted capabilities, changes in Definition will then necessitate revision of efforts in these phases as well.

To summarize, Definition is the most crucial technical objective in determining the course of a knowledge engineering effort. Our conclusion is that several factors preclude an initial viable definition of capabilities. This limitation is one major reason that knowledge engineering is highly variable and iterative in the way it pursues its objectives. The most salient factor is the knowledge gap among the participants, which inhibits the communication required to define the desired expert system. We see no purely technical approach to reducing the knowledge gap more rapidly. Another important factor is the heuristic nature of the KE's knowledge for system design and implementation, which may be inherent or may simply reflect the present immaturity of the emerging discipline of knowledge engineering.

B. Knowledge Acquisition Objective

Subobjectives

Identify knowledge categories required by targeted system capabilities
Elicit domain knowledge from DE
Informally structure knowledge

The Knowledge Acquisition objective refers to the accumulation of the specific knowledge required to support implementation of the targeted system capabilities. As discussed above, domain knowledge is also elicited in pursuing other objectives of the knowledge engineering process. In all cases, the activities involved in collecting domain knowledge may be nearly identical: interviewing the DE, observing the DE, reading printed descriptions, etc. However, depending on the technical objective that is the KE's current focus, the form and sequence of questions and the organization of answers, observations, and notes can differ. Roughly speaking, the KE is more interested in the "what" (the goals) of the DE's performance when defining system objectives and in the "how" (the methods) when pursuing Knowledge Acquisition. The specific knowledge that needs to be acquired is determined by what capabilities have been targeted. The KE's ability to characterize the nature and scope of that knowledge and to develop a plan for acquiring it efficiently depends on his knowledge of prior attempts to build similar systems. At the present time, that knowledge can provide guidance but not detailed procedures for the KE to follow (see, for example, Buchanan, et al. [1983]). The prior case studies available to the KE are limited both in number and in the types of tasks considered. The formulation of a rough mapping between task characteristics, expert system capabilities, and the

knowledge and techniques needed to achieve those capabilities is just now becoming possible (see, Stefik et al. [1983], for one attempt to specify this mapping). Therefore, even the best-informed KE cannot specify a priori the questions and observations that will provide the knowledge needed to implement a capability, even one similar to that in another operational system.

The knowledge gap between the KE and DE that affects Definition also affects Knowledge Acquisition. The KE initially does not have the domain-specific semantic and pragmatic knowledge needed to evaluate and organize the DE's statements or to direct interactions into related or important areas of knowledge. The DE doesn't understand the KE's technical objectives and methods and may have difficulty articulating his knowledge; thus, he is rarely in a position to assume initiative. Therefore, to manage Knowledge Acquisition interactions, the KE must depend on relatively domain-independent syntactic knowledge, based on a general framework for viewing tasks of this type (e.g., interpretation, diagnosis, etc.) and later based on the representation language he selects.

General frameworks for expressing task descriptions provide a few fundamental concepts to describe and relate goals, procedures, and data. They provide a simple uniform syntax for structuring Knowledge Acquisition interactions. The most typical method is top-down progressive expansion of detail. Knowledge acquisition concerning goals involves characterizing their dependencies: relative priorities, enabling relations, and constraining relations. Such characterizations frequently include judgments of degree of belief or certainty. Procedures are characterized in terms of their enabling and triggering conditions, the more primitive actions they integrate, the resources they require, and their effectiveness. Data are characterized by their source, their application, and often judgments of reliability. In using a general framework for describing knowledge the KE relies upon ad hoc natural language, if-then rules, and diagrams for representation.

Once the KE has selected the representation language(s) the system will use (part of the Formalization objective), that language can be used to guide Knowledge Acquisition. Representation languages provide a more detailed syntax and general semantics for describing goals, procedures, and data. They allow the KE to pose more precise requests for knowledge than do general frameworks. At the same time, selection of a representation is a commitment that excludes some domain knowledge from the KE's consideration and usually changes how the KE pursues his several objectives. From that point, the activities for acquiring knowledge and formalizing it are often tightly coupled, especially when the KE can also implement the knowledge base incrementally. Even with this coupling however, knowledge acquisition does not become a passive, mechanical process for the KE; active interpretation and integration are still required (Buchanan, 1981).

One problem with syntactically-driven approaches to structuring knowledge acquisition interactions is that they can confuse or irritate the DE because transitions and emphasis may not correspond to his intuitions about the structure of the domain knowledge. Until the KE becomes more familiar with the domain and can use semantic and pragmatic knowledge to guide elicitation, his options for overcoming the problems of syntactically-driven interactions are limited.

One method of improving dialogue management involves varying the focus of elicitation. At the least, the KE can employ some content-free heuristics to modulate repetitive patterns of interaction, which can easily result from strictly depth- or breadth-first, top-down exposition of task goals and methods. Depending on the modularity of the system's intended capabilities, the KE might also direct elicitation toward knowledge specific to each capability in turn, perhaps in order of importance or of planned implementation.

Alternatively, the KE can use syntactic methods in a bottom-up manner that gives the DE more initiative in controlling transitions and emphasis. The most prevalent method is to orient interactions around real or hypothetical task situations or cases. Using case generation, even a relatively inarticulate DE can usually share the initiative with the KE. Once the DE has described an "important" case and how it should be treated, the KE can use syntactic methods to extend and elicit the knowledge necessary to analyze that case. When appropriate, the KE can use structural features of a representation to prompt for related cases. For example, the KE might ask, "Is there a case in which the procedure you just described is omitted?", "Is there a case in which those two goals have the opposite priority?", or "Is there a case in which those data have different implications?". This type of prompting must be undertaken judiciously, since such questions may be misdirected and anomalous from the DE's perspective. Until he acquires some semantic and pragmatic knowledge about the domain, the KE's best tactic is to promote the DE's generation of interesting different cases. The KE can then build his knowledge of the domain by inferring the dimensions along which the DE chooses to elaborate and sequence cases.

There are at least three limitations of relying on interactions built around cases. First, multiple cases may contain much redundant knowledge. Each subsequent case therefore provides less new information, while the time required to discuss them may not decrease appreciably. It thus becomes less efficient over time. Second, the interaction over a case may cover many more considerations than required for the targeted capabilities. This is, of course, a function of how narrow the target set is in relation to the full task and of how much context is required to present a case meaningfully. Third, the focus of the particular cases generated by the DE may not be sensitive to any priorities or plans the KE has developed for working on different capabilities.

These difficulties imply that the KE's dialogue management plan for knowledge acquisition can emphasize case-oriented elicitation early in the process. As the knowledge base grows and other methods for effectively controlling focus become available, case discussions should be used less frequently. Skilled KEs generally rely on a combination of elicitation methods including (1) dialogues structured by applying general task description frameworks and formal representation languages and (2) bottom-up case-structured dialogues in which syntactic structures are used to derive and execute local plans for the interaction. However, the overall set of activities cannot be planned firmly. Instead, changes in interaction mode are triggered by the KE's perception of the effectiveness of the methods in use for cueing the DE and the rate at which new knowledge is being elicited. At present, there is no adequate model for how the KE makes these determinations.

Another aspect of the Knowledge Acquisition process worthy of note involves feedback opportunities for the KE and DE. When the process is undertaken as a series of protracted interactions, the KE may accumulate a body of knowledge and be uncertain as to its completeness, consistency, and ability to support the targeted capabilities. He cannot provide the DE with assurances about progress, yet such feedback is desirable for maintaining the DE's motivation. Coupling Implementation with Knowledge Acquisition (and Formalization) is a means for both the KE and DE to obtain feedback that may alter their approach to Knowledge Acquisition. When systems are built in development languages that facilitate the rapid coding and testing of prototypes, the ability to quickly produce and review initial results typically elicits enthusiasm and increased motivation from DEs. The requirement for feedback in Knowledge Acquisition is thus another reason knowledge engineering has typically been iterative and incremental.

To summarize, the pursuit of Knowledge Acquisition is closely related to the pursuit of other objectives. Definition determines what knowledge needs to be acquired. Formalization provides structure for managing knowledge acquisition interactions and for monitoring completeness and consistency of acquired knowledge. Implementation provides further feedback on completeness and consistency and enables a more goal-oriented pursuit of knowledge.

C. Formalization Objective

Subobjectives

Select a knowledge representation language
    Characterize structure of acquired knowledge
    Contrast knowledge characteristics with language features
Translate informal knowledge descriptions into formal representation language

The Formalization objective encompasses the activities that create a formal description of the knowledge acquired from the DE using a representation language. Ideally, the KE abstracts certain features of the acquired knowledge and selects (or designs) a congruent representation language. However, there are no objective criteria for determining an appropriate representation formalism for knowledge with given characteristics. Formalization is ad hoc with respect both to the definition of knowledge characteristics and what they imply for representation. Each KE uses guidelines based on his knowledge of previous knowledge engineering efforts and of knowledge representation research in general.

While representations have considerable similarity in their expressive power, they have idiosyncratic strengths and weaknesses. The weaknesses sometimes prove a representation unsuitable for easily expressing specific knowledge or for supporting satisfactory implementation of a system capability. Since there is no general capability for automatically

translating knowledge expressed in one representation to another,[4] choosing a representation that later demonstrates weaknesses can create problems for a knowledge engineering effort. Only very recently has there been any attempt to propose general principles for matching knowledge characteristics to representational capabilities (Buchanan, et al., 1983; Stefik, et al., 1983).

The principles that have been proposed for determining a representation language are derived from successes and failures of prior efforts rather than from theoretical analysis. Stefik et al. (1983) consider three broad aspects of the acquired knowledge: the hypothesis space, the problem-solving process, and the data. These correspond to the goals, procedures, and data of general task description frameworks. The important dimensions of the hypothesis space are its size and structure, where structure refers to dimensions, such as temporal, logical, and semantic, upon which the space can be decomposed and operated. The problem solving process is characterized with respect to its homogeneity and its dominant method (e.g., backward-chaining, forward-chaining). Data are characterized with respect to their certainty, completeness, and stability.

The guidelines presented by Stefik et al. (1983) are useful, but are only a first approximation to an understanding of how to select or design representations for an expert system. They are derived from only limited experience with the potential set of problems to which knowledge-based systems might be applied. Even within the range of problems that have been considered, the guidelines are not unique or precise prescriptions and cannot replace the knowledge a skilled KE uses to select a representation for use in a new system.

The development of practical knowledge about the strengths and weaknesses of different knowledge representations for different types of problems has been impaired, probably more than any other applied knowledge, by the fact that most efforts to develop expert systems have been dominated by the concerns of academic research. The rewards have been greater there for designing new representation languages and associated computational environments than for intensively analyzing the applicability of existing ones. Consequently, there has been a proliferation of representation languages and systems, few of which have been applied seriously to more than one or two problem domains or have been used outside the institution in which they were developed.

There is also a practical aspect to the result of coupling expert systems efforts to inventive research on knowledge representation. Few representation languages have been implemented in transportable, production-quality computational systems. Thus, unless he has the skill and resources to develop his own implementation, a practicing KE's choice of languages is limited. If

---

[4]Translating between representations is a problem similar to translating between conventional algebraic programming languages. Despite great interest, little progress has occurred on the latter problem and there is little reason to believe that the capability will be achieved first for the more complex AI representation languages.

he does not, then a priori constraints on the hardware and software environment (e.g., as imposed by the customer) can further limit his options. When the choice of representation systems is known to be limited, these constraints are best evaluated during initial efforts on Definition. Doing so can allow the KE to use knowledge of hardware or software limitations to circumscribe the system's capabilities and the types of knowledge the system will require.

Once a representation language has been selected or designed, the KE can use it to codify informally represented knowledge elicited from the DE. This is not necessarily straightforward, since there may be alternative ways of expressing the same knowledge within a representation. A common uncertainty is the granularity with which knowledge is to be described; that is, what knowledge will be considered primitive and what will be composite. The KE must determine granularity such that the representation of knowledge is transparent. Transparency means that the expression of the knowledge should not be excessively decomposed or contorted from the DE's natural way of expressing it. Transparency is important in facilitating debugging and maintenance of a system and for making the system's behavior more understandable to its users. In particular, it helps make explanation capabilities in the system congruent with users' conceptions of an appropriate level of abstraction for understanding problems in the task domain.

When the KE has selected the representation language and has begun to encode some knowledge into it, he can couple more closely the activities of acquiring and encoding knowledge, bypassing intermediate informal descriptions. As described above, this provides mechanisms for dialogue management using the syntax of the representation. The KE must still, however, interpret and integrate the knowledge elicited from the DE.

D. Design Objective

Subobjectives

Develop system software architecture

Select system hardware architecture

The Design Objective entails formulating the hardware and software architecture of the system. The process of producing a design interacts with and depends on activities in the Definition, Formalization, Implementation, and Testing Phases. The resulting design also depends on compatibility, cost, or equipment constraints imposed by the customer. Constraints on hardware and software options may limit Design options and even affect the capabilities that can be incorporated into the system. In the absence of strong customer constraints, the KE can design a software and hardware architecture based on the most effective match between target capabilities and hardware/software tools and techniques to realize the capabilities. If he cannot customize a representation language, he will be limited to architectures that can be realized in an existing production-quality language system. In either case, the limited number of prior cases from which the KE can draw can create uncertainty about the applicability of any particular design to the current problem and the expected performance of the system when implemented. Prior to implementation, the type and scope of knowledge obtained during Knowledge

Acquisition will suggest to the KE the applicability of the various standard software design options available (e.g., type of inference system, structure of the knowledge base, representation of problem-solving strategies, explanation facilities) and hardware options available (e.g., graphics display vs. character display, keyboard input vs. mouse input, dedicated processor vs. time-shared system).

A major task in system design is the integration of the representation language system that encodes the task expertise with satisfactory user interface software and hardware to provide the system's interaction capabilities. If the KE is constrained to use a particular software system, then the representation is a given. Otherwise, he must use basic computer science skills and AI knowledge to generate a design that will be both adequate and efficient. While existing development systems from which the KE may select his programming environment offer user interface facilities, most are tailored to the needs of a researcher with AI programming skills rather than a customer or DE. Hence, in developing a finished system for a commercial or military application, the KE will most likely face a significant design effort for the user interface features.

Most of the high-level guidelines and knowledge to be used in interface design (e.g., partitioning of initiative) should have been garnered during Definition through discussion among the participants. Lower-level design considerations (e.g., formats, dialogue management, heuristics, etc.) may be specified during initial Definition, but more likely will emerge during Knowledge Acquisition and Formalization. These phases will specify in detail the type and format of knowledge to be acquired from the user or provided by the system to the user, as well as the structure imposed on that knowledge. This specification will enable the design of interaction protocols from which detailed user interface features and characteristics can be developed. However, interface design for knowledge-based systems (or computer systems in general) is not a mature discipline; again, the knowledge the KE can bring to bear is incomplete and heuristic. In most cases, testing by the DE and initial use by the intended user population are required to revise the design and implementation of the interface--another reason for the reliance on incremental system development. In the expert systems field it is typical to defer low-level interface design and implementation until the implementation of the fundamental problem-solving capabilities is relatively robust and ready for use outside the development setting.

E. Implementation Objective

Subobjectives

Implement inferential capabilities and user interface
Encode domain knowledge
Detect and debug programming errors

Implementation entails coding the formalized knowledge acquired from the DE in the selected hardware-software environment and verifying that the system runs without software errors. (Such testing is distinguished from that undertaken during the Testing phase which assesses the system's effectiveness in realizing the target capabilities.) The Implementation process provides the KE with information that can require additional work and revision in the

Definition, Knowledge Acquisition, Formalization, and Design phases. This information bears upon decisions that prior to implementation were based on incomplete and uncertain knowledge generalized from prior system development efforts. The utility of such information is a major reason that Implementation has been undertaken incrementally, concurrent with the pursuit of other objectives, in many efforts to build knowledge-based systems. A general principle for knowledge engineering urged upon would-be KE's (Buchanan, et al., 1983) is the early implementation of an experimental prototype.

The extent to which the KE must code the basic software environment, as opposed to just the formalized domain knowledge, depends on the representation language he has selected. User interface software must generally be developed even if the KE can adopt an existing language system, but its design and implementation are usually deferred until the system's problem-solving capabilities have been implemented and tested. Thus, at worst, if he is implementing his own basic software environment, the KE must only implement the representation language, the associated problem-solving mechanisms, and a skeletal interface before starting to encode the formalized domain knowledge. However, experience indicates that including specialized editors and debugging tools in the software environment greatly enhances the efficiency of incremental implementation efforts.

Of course, the KE's approach to implementation also depends upon the availability of the DE. Ready access to the DE supports an incremental approach, especially where an early selection of representation language can be made. Limited access--intensive, infrequent meetings spaced over time--reduces the opportunity for incremental implementation and thus, given the current state-of-the-art in knowledge engineering, the effectiveness of the system-building effort. Except in cases where the KE has been able to adopt a complete existing architecture for a new problem domain, successful efforts have included a DE as a regular member of the system-building team for the full duration of the project effort.

Incremental implementation and the concurrent pursuit of other objectives is inevitable, even desirable, as long as the KE's knowledge is as unsystematic, incomplete, and uncertain as it is at present. The incremental approach provides the KE with rapid feedback that can provide early indications of the advisability of revising system design decisions. The partial working system also serves an organizational function in helping to maintain and promote the motivation of the system building team.

F. Testing Objective

Subobjectives

Formulate diagnostic test cases
Assess system performance on test cases
Collect data on user's interactions with the system
Determine system performance on other targeted capabilities
Provide information required for effective revisions

Once a meaningful subset of the formalized domain knowledge is implemented, the KE can pursue Testing of its targeted problem-solving

capabilities. The typical method of testing is for the KE and DE to define critical (i.e., topical, important, frequent, or difficult) problem situations for which the DE can specify criterial performance. Normally, testing with these cases is repeated at points during the growth of the knowledge base and whenever the architecture or implementation of problem-solving capabilities is otherwise modified, as a check for new conflicts and inconsistencies that may have been introduced.

A second aspect of Testing involves evaluation of user interface capabilities. While these can be tuned initially by the development team, use by others from the intended user community is required. This testing has typically been limited and informal in many prior efforts conducted in research environments.

The results of Testing can bear upon decisions and knowledge accumulated in pursuit of each of the other objectives of the system-building effort. They may lead the KE to determine that:

a. Particular capabilities are inappropriate or intractable and should be eliminated, perhaps to be replaced by others.

b. The domain knowledge is incomplete, inconsistent, or inaccurate.

c. The representation language lacks the required expressive power or transparency.

d. The hardware-software architecture lacks sufficient capacity to run the system or speed to execute it at an acceptable rate.

e. The problem-solving methods and strategies are inadequate--that is, the system's behavior is not "expert."

Unsatisfactory performance on criterial cases might be a symptom of any of these problems. Initially, the KE responds by iterating on Knowledge Acquisition and Implementation, since these are most accessible to revision. Eventually, however, if problems persist or if new problems continue to emerge, the KE need to consider modification of the Definition, Formalization, or Design objectives. Such modifications may have more far-reaching ramifications for the nature of the resulting system. When system developers opt for new capabilities, representation languages, or architecture, the domain knowledge base usually requires fundamental changes that entail total redesign and reimplementation of the initial prototype (Buchanan, et al., 1983).

This process of test and revision is yet another reason for the desirability of incremental design, implementation, and test. By testing performance as the system evolves, it may be possible to accelerate the point where reimplementation, if it is required, can occur prior to a large investment of time and resources on the initial prototype development effort.

SECTION III

CURRENT TECHNOLOGIES FOR ASSISTING KNOWLEDGE ENGINEERING

The model of the knowledge engineering process presented in the previous section provides a framework for consideration of alternative approaches to automating portions of the KE's tasks. Any new proposals and designs for such automated systems must consider previous and current attempts to provide assistance to DEs and KEs during system development. This section selectively reviews this body of research.

Two general approaches have been adopted to facilitate the knowledge engineering process (Barr and Feigenbaum, 1982). The first seeks to facilitate the interactive transfer of expertise (ITE), those phases of the knowledge engineering effort during which the KE elicits, formalizes, and encodes the DE's relevant domain knowledge. Some systems following this approach aim to assist the dialogue management, bookkeeping, and translation performed by the KE. These systems "interview" the user to collect domain knowledge. Other systems provide high-level programming languages specially designed to capture and represent heuristic, rule-based expert knowledge. In other cases, attempts have been made to develop knowledge base checking facilities for evaluating the consistency and completeness of the encoded knowledge. Finally, some work has been directed toward supporting the definition of system capabilities and the formulation of a software design. Development of such aids in many cases has evolved in the context of a larger project to implement a particular expert system; hence, such projects have often had a pragmatic, ad hoc nature.

The second approach to facilitating development of expert systems is "automatic theory formation" or, simply, automated learning. That research aims to develop systems that induce new knowledge from experience--for example, by analyzing the behavior of the DE on particular test cases. (The Meta-Dendral System [Lindsay, et al., 1980] illustrates this second approach). These systems would replace the ITE process as a method for developing knowledge bases and would apply to problems in which ITE is not productive, either because there is no consensus among DEs or because the DE cannot articulate his knowledge. Work on automated theory formation has typically been more theoretical and to a large extent has been pursued independently from practical applications of expert systems. Because its applied (as opposed to theoretical) focus matches the concerns of the present project, our review considers only those efforts directed at ITE.

The developers of the first expert systems quickly faced the problems of managing the growth of their system and of providing facilities for end users to control and maintain them. KEs found that once a representation language and system architecture had been selected, the task of eliciting and encoding new knowledge was more systematic, albeit still difficult, using generic tools provided within AI programming systems. They responded by developing specializations of these tools--knowledge structure editors, break and trace packages for debugging, file managers--geared to the representations and capabilities of their expert system. In so doing, they endeavored to provide the means for altering the expert system using the concepts in the application domain instead of the programming primitives that implemented that

application.

Following the early research on ITE conducted in specific application domains, a second generation of efforts has attempted to extend the initial work and develop generic tools for constructing expert systems. Such tools have taken the form of either skeletal systems or specialized programming languages. The skeletal systems provide a fixed problem-solving architecture (knowledge base representations and control mechanisms) and user-support modules for building knowledge bases using the system. Use of such skeletal systems is limited to problem domains for which their capabilities are congruent with the customer's needs and their representations adequate to capture the structure and content of the domain knowledge. The specialized programming languages provide data and control constructs better tailored to the knowledge structures used in expert systems than the constructs of general AI languages such as LISP. These languages are more general than skeletal systems, enabling more flexible selection of architectures and capabilities. However, they require the user to design representations and an overall architecture and consequently to engage in greater implementation effort.

In general, skeletal systems aid the KE in achieving the objectives of Knowledge Acquisition, Implementation, and Testing, while they eliminate Design effort. Specialized programming languages support Implementation directly and Design and Testing indirectly through the transparent high-level constructs they provide to the KE.

All the efforts to aid ITE have been oriented toward either KEs or DEs with considerable sophistication about the expert system they are helping to develop or maintain. Use of ITE aids by DEs with no KE support has been achieved only for relatively mature systems in which the user's role is to update and extend an already extensive knowledge base. The use of skeletal systems for building a new knowledge base and the use of specialized programming languages remains limited to KEs. However, the availability of such systems and languages reduces the level of programming ability a KE needs to implement an expert system.

The following discussion of representative systems is divided into two parts. The first describes systems in which support for ITE was developed for a particular architecture and problem domain. The second describes more general efforts embodied in skeletal systems, specialized programming languages, and other general systems for implementing task expertise.

APPLICATION-SPECIFIC ASSISTANCE FOR ITE

TEIRESIAS. TEIRESIAS (Davis, 1977) was developed as a subsystem to support the growth and maintenance of the MYCIN system (Shortliffe, 1976) knowledge base. Its facilities include tools for modification and debugging of the production rules MYCIN uses to encode knowledge about the relationships between clinical data, infectious diseases, and drug therapies.

TEIRESIAS' mode of interaction is a mixed-initiative dialogue using a mixture of quasi-English language and menu interactions with the user. The dialogue model casts the user in the role of a teacher who is instructing the system about new domain rules. The system assumes the role of a motivated

learner, raising questions of clarification about what it is told based on the relationship of the new rules to old ones it knows. For example, it will point out to the user that a new rule does not contain a predicate or action clause that occurs in other rules referencing the same concepts.

TEIRESIAS' mechanisms depend on knowledge it has about concept categories and rule structure. This knowledge allows it to parse quasi-English rule descriptions into its internal representation. It also permits TEIRESIAS to raise questions about the semantics and pragmatics of proposed rule modifications or additions. This application-specific meta-knowledge was initially programmed into TEIRESIAS, but it has mechanisms for modifying the knowledge. For example, knowledge of concepts frequently referenced together is induced dynamically as the knowledge base changes.

To support debugging, TEIRESIAS interfaces to MYCIN's explanation facilities and case library. When modifying or adding rules, the user can ask for a dynamic trace of rule tests to determine why specific conclusions were or weren't reached. This aids the user in revising existing rules when new rules are added, so that undesirable interactions among rules can be corrected as the knowledge base grows.

TEIRESIAS' approach is based on two fundamental assumptions. The first assumption is that the host system's (i.e., MYCIN's) control structure and knowledge representations are understandable abstractions of the domain for its users. TEIRESIAS' users were the same KEs and DEs who had built MYCIN originally. They were therefore familiar with MYCIN's abstractions of the problem domain and problem-solving process and could think in terms of them, even if they were not completely intuitive. TEIRESIAS would not be accessible to use by others without considerable training about how domain knowledge is represented and used by MYCIN.

TEIRESIAS' use of meta-knowledge about the host system's implementation reflects a second assumption: that MYCIN's control structure and knowledge representations were a sound approach and therefore stable. The architecture and interface of the ITE functions are directly dependent on those of the host system and would require revision if the latter were modified. MYCIN was in fact a relatively mature system, with a significant existing knowledge base, at the time TEIRESIAS was designed and implemented. The value of implementing ITE support for a mature system depends on whether the knowledge in the application domain is expected to change over time, on how complex the "manual" process of modifying the knowledge base is, and on the skills of the individuals who are available to implement modifications.

KAS. The Knowledge Acquisition System (KAS) (Reboh, 1981) was designed as an ITE aid for the PROSPECTOR system (Duda, et al., 1978). Like TEIRESIAS, KAS was designed and implemented only after its host system's architecture was stable and a substantial knowledge base existed. Similarly, its architecture and interface are dependent on the representations and control structures of PROSPECTOR. However, it does not use application dependent meta-knowledge about the PROSPECTOR problem domain (i.e., interpretation of geological data). Thus, KAS is transportable with the bare PROSPECTOR architecture (i.e., as a skeletal system) to other applications.

KAS takes advantage of PROSPECTOR's network knowledge representation for guiding the acquisition of related rules. In particular, it supports the joint efforts of a KE and DE to define a new inference model, which describes the rules relating geological data to a specific type of ore deposit. PROSPECTOR uses networks to partition and structure its rules into inference models, and KAS uses the structural syntax of the network representation to stimulate and monitor the elicitation of models. In addition to supporting knowledge acquisition in this manner, KAS provides the KE with capabilities for debugging and evaluating the performance of models by tracing their execution on stored cases and new ones provided by the DE.

KAS's basic approach to model building is initial top-down definition of the abstractions in an inference model followed by iterative top-down elaboration of the arcs and nodes of the network created by the original definition. These elaborations include the logical semantics of the arcs linking nodes, the numeric values of parameters to be tested against data and to be used to assign confidence to inferences, and specifications for the format of questions and answers PROSPECTOR should use when asking its user about parameters.

The main subsystem of KAS used in model building is the REsident Network Editor (RENE), which incorporates knowledge about PROSPECTOR's representational and computational formalisms. RENE provides an interactive editor for networks, automatic bookkeeping for the model implementation process, dialogue management during elicitation, and an interface for controlled execution of inference networks.

The structure editor, like those implemented for AI programming languages, provides the user with primitives for manipulating and altering complex data structures--in this case, networks. Commands for changing the editor's attention and for modifying knowledge reference arcs and nodes rather than the more primitive data types (e.g., strings of characters, lists, atoms) used in PROSPECTOR to implement these abstractions. The goal is both to prevent syntactic errors that could occur if the user manipulated low-level representations directly and to allow the user to manipulate constructs corresponding to those he uses to articulate his expertise during problem solving.

The bookkeeper is an "intelligent agent" that "looks over the shoulder of the KE" as he interacts with RENE's other facilities. It prevents syntactic errors--for example, by noticing that the KE hasn't connected parts of an inference network. It fills in defaults for descriptive and quantitative attributes of arcs and nodes that the KE does not supply. It alerts the KE to ramifications of inputs for prior inputs. The bookkeeper uses a hierarchy of knowledge about representations, about the host system, and, to the extent that it has been supplied, about the domain. This knowledge is used by a data-centered programming model that allows a set of actions to be associated with each type of element of a representation. These actions (e.g., generating a default value) are executed whenever an element is added or modified. In this way, RENE generates side-effects for actions initiated by the user that reflect the system's built-in knowledge of possible entailments of those actions.

KAS primarily employs a question-and-answer dialogue protocol, generatin-

its output from pre-stored templates. It uses menus and a quasi-English command language for user inputs. To inhibit user errors it uses the grammar of the command language to prompt the user rather than attempting to parse full statements. The grammar interfaces to PROSPECTOR's taxonomic networks (networks that represent the concepts used in the inference networks) to extend the lexical richness of the command language as the knowledge base grows.

During model development, RENE's capabilities for controlled execution of the evolving model within PROSPECTOR allow immediate incremental testing and refinement. Besides helping to prevent the propagation of dependent errors, the independent debugging of parts of a model provides motivating feedback to the KE and DE.

Once the user has developed an initial model, KAS can be used to install and tune it within the existing PROSPECTOR knowledge base. Tuning includes the addition of model-specific elaborations that modify defaults supplied by RENE, force specific question sequences, and control the focus of attention during a consultation that accesses the model. At this stage, KAS also offers facilities for "batch mode" testing of the complete model on cases supplied by the expert. These facilities provide automatic variation of answers stored for the cases to allow a sensitivity analysis. They also allow the mixing of stored answers with those acquired interactively so that the KE and DE can more easily explore a particular issue in variety of contexts.

The developers of KAS have drawn several general conclusions from their effort. First, regarding the knowledge that must be acquired, support for IKE should extend beyond the acquisition of domain problem-solving knowledge. Effective interfaces for knowledge-based systems like PROSPECTOR require domain-specific meta-knowledge to provide dialogue control if consultations are to follow a course that is acceptable to its users. KAS is a major aid for adding and testing such meta-knowledge for new inference models. Second, use of an existing knowledge base has important benefits for supporting the acquisition of new knowledge. In particular, experience with KAS has shown that expanding PROSPECTOR's network of taxonomies without regard to its actual use by existing inference models is an aid to acquiring new models. This benefit derives from the use of the taxonomies to dynamically expand the vocabulary of the command language. The knowledge a system already contains is also important for supporting the understanding of incomplete references by the user. Without this capability, unless the system can somehow provide enough context for the user to keep track of the contents of the knowledge base, it is difficult for the user to interact with a knowledge acquisition aid.

The final conclusion concerns the generality of knowledge acquisition aids. Effective aids require knowledge about the representations used in their host systems. The details of such representations depend on the problem domain and purpose encompassed by the host system. Thus, the generality of specific aids can never be greater than the generality of their host system. PROSPECTOR/KAS attempts to maximize their generality through a "layered" architecture that localizes the modifications needed to adapt the system to different domains and purposes. Several attempts to adapt the architecture to different classification and interpretation problems have confirmed that minimal changes are required to support applications of PROSPECTOR/KAS to

other members of the class of data interpretation problems.

THE ONCOCIN KNOWLEDGE BASE VERIFIER. ONCOCIN is an expert system that provides its users with recommendations about cancer treatment therapies for individual cases. Part of the ONCOCIN system (Suwa, Scott, and Shortliffe, 1982) is an ITE aid for the extension and revision of the ONCOCIN knowledge base. This aid checks the completeness and consistency of the knowledge base whenever it is modified. Unlike the approach taken in TEIRESIAS and KAS, the ONCOCIN knowledge base verifier allows testing for knowledge consistency and completeness prior to the existence of or access to a functioning host system.

ONCOCIN rules determine a therapy protocol represented as a list of "action parameter values". Each rule includes a context and a set of conditions for determining its applicability to a case. The verifier operates by syntactically analyzing rules that recommend the same action in the same context and generating a table whose row entries are all the possible combinations of condition parameters found in the rules. The table indicates:

a. Missing rules: combinations of condition parameters not associated with any therapy protocol.

b. Conflict: combinations of condition parameters that would succeed in the same situation and are associated with different protocols.

c. Redundancy and Subsumption: combinations of condition parameters that would succeed in the same situation and are associated with identical protocols.

Only conflicts are true errors, but the other data serve as additional foci for a review and revision of the knowledge base by the KE and DE. The table display constitutes the entire user interface of the knowledge base verifier. This limited interface restricts the verifier's utility to a KE who has a deep understanding of its operation.

The verifier was initially developed and used to support the construction of the original ONCOCIN knowledge base. It is specific to ONCOCIN's rule representation, but independent of the domain semantics and system architecture. The generality of the approach does seem limited by domain characteristics, however. To avoid a combinatorial explosion in the size of the table, the KE must partition rules by "context" into sets sharing a small number of condition parameters. For the discovery of missing rules to be a useful activity, most combinations of condition parameters must be meaningful in the problem domain.

SECS. SECS (Wipke, et al., 1977) is an expert system that helps its users solve organic chemistry synthesis problems. Its knowledge base is a body of rules, called transforms, each of which relates a target substructure to its precursors in a reaction. SECS includes an ITE support facility that allows its DE users to maintain and extend its knowledge base.

Modifications and additions to the SECS knowledge base are made using the

ALCHEM language, a quasi-English command language with a rigid syntax. The ALCHEM commands enable the description and manipulation of transforms in terms of schema-like abstractions. Each "slot" of the schema has its own specialized syntax which is recognized by ALCHEM. Although the ALCHEM syntax is rigid, the ITE interface is flexible in that the user has full control over how to proceed in defining a new transform schema.

SECS and ALCHEM represent an extreme approach for using domain and system-dependent knowledge in implementing support for ITE. At the cost of generality and of considerable commitment by its users to master new skills, the result has been a mature system which is now completely maintained by its user community.

KOL. The Knowledgeable Opponent Librarian (KOL) (Alperovitch, 1982) is an ITE aid developed for use with a knowledge-based tactical gaming system. It supports the elicitation of tactical plans that can be executed by an automated opponent in the host system.

KOL's elicitation method is based on top-down refinement. The user, either a KE or trained DE, is prompted to describe a tactical plan at progressively lower levels of abstraction. Each refinement expands an action into a number of more specific actions. Ultimately, the refinement must reach a level where the descriptions reference only a predefined set of domain-specific primitive elements.

KOL allows the user to exercise initiative and return to previously expanded nodes in the plan tree in order to enter modifications. It has bookkeeping facilities for informing the user when such modifications have ramifications for other nodes and for monitoring whether all branches of the plan have been expanded to the level of domain primitives.

The KOL user interface uses stored text templates to generate prompts. It has a structure editor for input that limits syntactic errors. It makes use of CRT display features to present context information during editing and to display plans. These features are also used during elicitation to provide limited context around the plan node being expanded. The interface does not provide capabilities for using existing plans to build new ones. In addition, because it can only interpret the design descriptions at the primitive level of the plan tree (i.e., the leaves of the tree), it can provide no assistance in evaluating the overall design.

KOL is essentially an aid for program design and automatic programming in a particular domain--that of submarine tactics. It forces the user to design a plan (program) specification in a structured, top-down fashion. It then uses an interpreter to translate the lowest level of the design, specified in a predefined abstract language, into Pascal code that can be executed by the host system.

The KOL approach to supporting ITE, like most of the others described thus far, is viable only after its host system is relatively mature. It depends on the existence of a stable domain-dependent set of primitive concepts and a language for referencing them. The KOL implementation is relatively modular, allowing the elicitation and bookkeeping facilities to be

used with different domain-specific primitive plan languages.

KOL's generality is limited to problem domains in which plans are conceived as hierarchical and in which the actions at each level of the hierarchy are sequential and independent. This approach to planning may be overly restrictive in certain situations and not well matched to humans' planning style. For example, for some problems the plans may not be most naturally represented as hierarchical structures. In cases in which the plans are hierarchical, actions in the plan may include concurrent and asynchronous actions. In addition, planners do not always develop their plans in strictly top-down fashion. Rather, they may generate plan elements opportunistically as decisions and choices at different levels of abstraction occur to them. Therefore, an aid for ITE should provide flexibility and robustness in the representation structure of the knowledge to be elicited and in the degree of user control over what portion of the structure to elaborate next. The need for such flexibility increases as the intended generality and intended scope of applicability of the aid for ITE increases.

ITE SUPPORT IN APPLICATION-INDEPENDENT KNOWLEDGE-ENGINEERING TOOLS

EMYCIN. EMYCIN (Essential MYCIN) (van Melle, 1979) is a skeletal system derived from the architecture of the MYCIN system. It has been applied to classification and interpretation problems in several domains.

EMYCIN contains all the support facilities for ITE that TEIRESIAS provided for MYCIN, except those that were directly dependent on domain knowledge programmed into TEIRESIAS or available to it from MYCIN. Thus, EMYCIN does not support the same quasi-English rule specification nor the checks for rule consistency that were features of TEIRESIAS's interaction with the user. Instead, EMYCIN has a high-level structure editor for modifying its knowledge base. Within the editor, rule specifications are entered in the LISP programming language. The editor provides some syntactic and lexical error checking and correction as well as automatic bookkeeping for the knowledge base. EMYCIN thus requires greater computer-related skills from its users than did TEIRESIAS.

The facilities carried over from TEIRESIAS include:

a. Interactive tracing and debugging of the reasoning leading to conclusions.

b. An interface to the performance system's explanation facility to obtain summary explanations of reasoning.

c. Automatic testing and comparison of stored case data and results against those produced by a modified knowledge base.

In general, ITE support in EMYCIN is more passive than that in TEIRESIAS, placing most of the responsibility for initiative in the user's hands.

ROGET. ROGET (Bennett, 1983) is an ongoing, experimental effort to provide

extensive support for ITE in the EMYCIN system. ROGET is unusual in that it is intended to consult with the user about how to design an expert system for his domain. ROGET applies meta-knowledge about how automated consultants for other problem domains have been implemented within the EMYCIN architecture. This experience is encoded both as specific facts about the other implementations and as generalized concepts and rules inferred from the specific facts. ROGET thus embodies a characterization of the class of problems to which EMYCIN is applicable.

ROGET applies a set of dialogue management rules to control a question and answer dialogue with its user. The dialogue guides the user through several phases of description of the desired system. The initial questions ask about the purpose of the system, the general features of the knowledge it must embody, and the data it will operate on. Examples from EMYCIN and other expert systems are used to illustrate the classifications the user can specify. Using these data, ROGET generates a probabilistic conclusion about whether EMYCIN is suited to the task and, if so, what time and resource expenditures the user should expect. It is able to use a trace of its own rules to explain its conclusions to the user.

In the next phase of the dialogue, ROGET guides the user in formulating the abstract conceptual structure of the task. Again, it uses its knowledge of other systems to illustrate possible answers. A conceptual structure, analogous to the inference models of PROSPECTOR/KAS, describes a generic model of problem solving for a class of tasks (e.g., clinical classification). ROGET uses the abstract conceptual structure it elicits to guide a dialogue elaborating specific models for the user's domain. This corresponds to the elicitation of taxonomic and inference networks in PROSPECTOR/KAS.

The features of ROGET are only partially implemented and there are no real data on the effectiveness of the system. Even so, ROGET is a significant new step toward support of ITE. It is the first system to provide assistance in initial problem formulation (part of the Definition objective) that is fully integrated with elaboration and refinement of the formulation. It does this using its knowledge about the host system and about the characteristics of specific types of problem domains implemented in the host system. ROGET is oriented toward use by a KE; however, once the initial characterization and conceptual structure have been entered by the KE (based on interactions with a DE), it should be possible for a trained DE to use the system to further elaborate the structure with only intermittent assistance from the KE.

The approach taken in ROGET suggests that a system providing assistance in early knowledge acquisition requires historical and pragmatic knowledge about its own structure and about the class of problems that it can address.

EXPERT. EXPERT (Weiss and Kulikowski, 1979), like EMYCIN and PROSPECTOR/KAS, is a skeletal system for building consultation systems that help solve classification problems.

EXPERT's support for ITE is limited to testing and debugging new knowledge. Adding new knowledge is a programming-like activity that occurs with no direct interface to the performance system. Instead the user must employ a standard text editor to describe the knowledge in a rigid, special

purpose language. The text file is then submitted to a compiler which detects syntax errors and generates an executable internal representation of the knowledge base. This base then can be utilized by the EXPERT performance system. Knowledge acquisition therefore requires users trained both on the system's problem-solving abstractions and use of general-purpose computer systems.

Testing and debugging facilities permit tracing the operation of rules on a case and comparing the results produced by different versions of a knowledge base over a set of cases. The emphasis is on design of inference models by debugging and refinement, with initiative strongly vested with the user. However, the need to use an external editor and operating system facilities to manage knowledge base maintenance assigns greater incidental bookkeeping responsibilities to the user than is common in similar skeletal systems.

AGE. AGE ("Attempt to GEneralize") (Nii and Aiello, 1979) is a system for designing and building expert systems. It bridges the gap between skeletal systems like EMYCIN and specialized AI programming languages. AGE aids its user in selecting and configuring control formalisms for the expert system the user intends to build. Ultimately, the user must encode the domain knowledge into the selected formalisms using the LISP programming language. However, AGE provides a software library comprising a rule interpreter, rule tracer, and explanation modules as well as the representation formalisms. The user has access to these through an intelligent front-end that performs automatic bookkeeping and provides active guidance during the design and implementation of the user's system.

AGE conducts a dialogue generated from templates to elicit a design from the user. The user predominantly communicates through menu choices, with text input limited to labels the user provides for components of the design. Dialogue management is based on prestored control knowledge about sequences of architectural decisions and implementation activities that realize the alternative architectures. At decision points, AGE offers the user choices and can provide advice based on its control knowledge. Using a command language, the user can escape from guided design elicitation to work on the design and implementation in any order he wishes. Subsequently, he can return to guided design, since AGE maintains a record of progress within both the guided and unguided modes.

Despite the user-friendly features of its front-end, AGE is strictly a tool for the knowledgeable KE. In addition to LISP programming ability, the user must learn to understand and use the technical terminology AGE uses to reference the software models it includes. ITE support is limited to design and implementation, with no facilities for assisting the KE in interacting with the DE to acquire domain knowledge.

GODDESS. The GODDESS system (Pearl, Leal, and Saleh, 1982) is an integrated knowledge acquisition and decision aiding system. The performance component of GODDESS is not an expert system, but we include the system in our discussion because it draws on AI research and emphasizes support for ITE.

GODDESS guides a naive computer user in the elaboration of considerations underlying an impending decision. The approach depends completely on

syntactic analysis of a formalized representation of the user's input accompanied by quantitative evaluations of preference and confidence judgments associated with elements in the representation. GODDESS can recommend a decision based on the user's input, but more importantly provides an interactive capability for the user to perform a sensitivity analysis--to determine the criticality of particular factors and values in determining the recommended decision.

GODDESS uses a simple generic representation of causal models for the user's input. This representation is an AND/OR graph of goals and actions derived from that used in the STRIPS (Fikes and Nilsson, 1971) system. Dialogue management is based on the structure of this representation. It starts with the user's ultimate goal and proceeds backward iteratively to elicit subgoals, alternative actions to achieve those subgoals, and preconditions for performing those actions. As each action is elaborated the user estimates its cost and its likelihood for causing its immediate goal to be achieved. The order in which subgoals (preconditions) are pursued in the dialogue is determined by GODDESS using user-supplied judgments of criticality.

The user interface in GODDESS is rigid and permits no user initiative. Since the system has no knowledge of any task domain and does not develop a hierarchy of primitive concepts, it must treat all user descriptions of goals and actions as unparsed character strings. This limits its ability to check the consistency of that information. The developers consciously sacrificed this capability for generality of applicability and availability to untrained users with a wide range of decision problem types.

ROSIE. ROSIE (Rule-Oriented System for Implementing Expertise) (Fain, et al., 1981) is a high-level programming language specially designed to support the implementation of expert systems. It has a relatively flexible English-like syntax and a variety of language constructs that support rule-based programming. However, it does not include an integrated design for knowledge representation and problem-solving control structures, such as the fixed design of EMYCIN or the alternative designs of AGE. Instead, the ROSIE user must construct these manually at his own initiative using the language's constructs. These constructs are intended to support transparent implementation of a variety of problem-solving architectures.

The ROSIE user interface includes the features AI languages such as LISP typically offer: interactive editing, interactive debugging using traces and break points, and automatic file management. However, these are oriented toward the level of abstraction of language constructs, not that of the user's design. Thus, ROSIE provides little direct support for ITE except insofar as it allows the user to program using the same concepts and English-like expression of rules as he uses to describe task expertise. ROSIE's appeal to the user is thus not its support for ITE, but in the flexibility it affords for implementing system capabilities not available in existing skeletal systems and in the ease it affords for expressing knowledge and rules.

CONCLUSIONS

The majority of efforts that have implemented knowledge-based systems for specific problem applications have included some attention to the support of ITE. In most cases, the support has evolved from that typically available in AI programming systems for interactive testing and debugging. The specializations of these facilities for knowledge-based systems include manipulation of more abstract representations of knowledge and control and of configurations of test data that are meaningful for the application. Support for the acquisition of new knowledge has been added to assist incremental extension and refinement of expert knowledge bases. TEIRESIAS and KAS have explicitly addressed knowledge acquisition support by taking an active role in the knowledge elicitation process. To do so, they use knowledge about the architecture of their host system, about specific features of the domain knowledge, and about characteristics of prior knowledge acquisition interactions.

Skeletal systems and other generic tools for building knowledge-based systems have included generalizations of the ITE support found in the application-specific systems. The ITE support in generic systems applies across most of the knowledge base development process. However, because it is based on the syntactic structure of skeletal system's representation formalism, active elicitation is more mechanical and less responsive.

To the present time, support for ITE has been oriented toward KEs and to a lesser extent DEs knowledgeable about formalisms of the host system. Thus, DE use has been limited to systems that are already stable and so are unlikely to require fundamental changes of capabilities, design, and implementation to accommodate extensions of their knowledge base. GODDESS, the one system described here that is oriented toward use by naive DEs, is not a true knowledge-based system in that it has no understanding of the knowledge the user encodes beyond its formal relational structure. For true knowledge-based systems, direct use by DEs for ITE appears possible only with special training. The amount of training required seems to be directly proportional to the complexity and flexibility of the system's representation formalisms and inversely proportional to the system's ability to apply domain-dependent knowledge in managing ITE interactions.

SECTION IV

INSTRUCTIONAL KNOWLEDGE ACQUISITION SYSTEM CONCEPT

In this section we present our concept of an automated knowledge acquisition system for elicitation of information from a DE for use in an instructional system. In deriving a concept for automating knowledge acquisition and reducing dependence on KEs for training system development, we considered the set of functions performed by the KE (as described in Section II). Our review of prior work on aids for ITE (described in Section III) provided data on the current state of the art in systems for ITE, the resources required to achieve various capabilities, and the feasibility of automating or assisting different aspects of the knowledge engineering process. Based on this analysis, we considered a variety of possible concepts for tools to assist the DE in pre-filtering ideas for automated training system applications, tools to assist the knowledge engineer in the system development effort, and tools to assist the elicitation of knowledge from the DE. The set of specific concepts we considered included:

a. An advisor to help the DE determine if a knowledge-based instructional system is feasible for his application

b. A tool recommender to advise the DE what training approach is tractable and what instructional technologies can be brought to bear on his application.

c. A knowledge engineering language tailored for the construction of surrogate instructor training systems.

d. A data base completeness checker to find redundancies, inconsistencies, and knowledge gaps in a wide range of application areas.

e. A test case generator for producing and managing tests of developed systems.

f. A system to elicit from the DE the basic declarative concepts in the domain and the relationships among them.

g. A system to elicit from the DE an expert task performance model for any domain (i.e., the goals and procedures used to perform the task correctly).

h. A system to elicit an expert task performance model for tasks in a particular domain class (e.g., tactics, radar operations) that would use class-specific knowledge to intelligently structure interactions with the user.

i. A system to elicit the deviations in an expert task performance model that would define the performance model (i.e., the model of the types of incorrect behaviors the trainee might choose in particular situations) used for skill diagnosis during training.

j. A system to elicit the rules used to diagnose trainee behavior during instruction (e.g., IF situation AND behavior THEN infer diagnosis).

k. A system to elicit training problems or situation descriptions that can be used by a generator to produce training problems.

To evaluate the promise of these concepts in meeting the objectives stated by NAVTRAEQUIPCEN, we developed a set of criteria for system utility against which these concepts could be weighed. The following paragraphs describe these criteria, and the subsequent discussion presents our concept of an automated instructional knowledge acquisition system (IKAS) that emerged from these considerations.

CRITERIA FOR EVALUATING IKAS CONCEPTS

EVIDENCE SUPPORTING FEASIBILITY. Perhaps the most fundamental criterion for evaluating the suitability of a concept is whether or not its implementation appears to be feasible, given current technology. Judgments of the feasibility of a concept depend on several types of evidence. Precedents from prior research indicate whether the knowledge and methods to perform the function under consideration can be sufficiently well described to support an implementation. Discussions with domain experts can indicate the extent of their knowledge, their ability to articulate it, and their ability to engage in particular types of human-machine dialogues about that knowledge. In general, the feasibility of realizing a particular IKAS architecture varies directly as a function of the scope and complexity of the concept.

EASE OF IMPLEMENTATION. A set of IKAS concepts judged feasible may vary on the anticipated time and effort required to implement them. Implementation time affects the cost of developing a system and therefore influences the evaluation of cost against expected benefit. A significant portion of development time for a specific IKAS depends on the time required to build and stabilize the host instructional system that the IKAS will serve. Another determinant is the scope and novelty of the targeted IKAS capabilities: the greater the anticipated difficulty in implementing a capability, the greater the expected implementation time. Novelty of a concept can be assessed by contrast to prior efforts at developing knowledge engineering aids. These prior efforts also provide a source of data for anticipating expected development times and resource requirements.

REDUCTION OF DE TIME-ON-TASK. One measure of the value of an IKAS is the extent to which it reduces the time DEs must invest to assist in the development of an instructional system. If an estimate can be made of the increased efficiency of the IKAS over manual knowledge acquisition, savings of DE time can be estimated by considering the relative proportion of overall system development activities involving the DE for which the IKAS will be used.

REDUCTION OF KE TIME-ON-TASK. The NAVTRAEQUIPCEN statement of work emphasized the inherent value of reduction of the KE's time and involvement in the system

development process. The cost and scarcity of skilled expert systems builders perhaps makes this criterion an important consideration in attempting to reduce costs. While the development of automated capabilities for acquiring instructional knowledge should reduce the requisite involvement of the KE on that portion of the system development tasks, the presence of the system will most likely entail other, new activities for the KE. For example, the KE might need to use the IKAS himself to review the results of DE-machine interactions and obtain knowledge he needs to perform other tasks. An understanding of the entire system-building process (see Section II) is critical in applying this criterion.

EVIDENCE FOR USER REQUIREMENT. This criterion reflects the extent to which an IKAS is perceived by potential users (both DEs and KEs) as necessary to supplement or replace manual methods. Even if the IKAS does not reduce time-on-task for either DE or KE, it may be required to alleviate the difficulties of scheduling face-to-face interactions between DE and KE. DE participation may not be possible unless it can be done on a spontaneous, opportunistic basis proximate to the DE work place. Inputs from DEs and system builders are instrumental for determining whether there are user requirements for an IKAS in order to build a knowledge-based instructional system for some domain class. Thus, when scheduling and geographic constraints restrict the interactions between KE and DE, the IKAS could serve as an institutional memory to record for subsequent review the interactions between the system and the DE or KE.

INCREASED FUNCTIONALITY. Aside from possible time savings and increased convenience, an IKAS may improve the quality of the knowledge obtained for the instructional system. It may accomplish this through the use of more systematic and thorough interviews than would be conducted manually. Relative functionality can be estimated by comparing the capabilities of the IKAS to those achieved manually, as evidenced by prior efforts. There is an obvious tradeoff with concept feasibility and time-for-implementation for approaches that attempt to significantly increase functionality.

GENERALITY AND SCOPE OF APPLICATION. Generality and scope refer to the breadth of applicability of the IKAS and to how much of the instruction typical of the domain can be accomplished via the instructional system. They also refer to how much of the knowledge used in the instructional system is obtained by the IKAS and to the extent of IKAS's role in the life-cycle of the instructional system. Ideally, when the domain knowledge is constantly changing, the IKAS should play a role in eliciting new knowledge from the DE to update to an existing system.

DE BACKGROUND AND SKILL REQUIREMENTS. A significant determinant in the success and utility of the IKAS will be the extent and type of knowledge (apart from domain-specific knowledge) the DE needs in order to use the IKAS. For example, it seems likely that an IKAS design will need to assume its user has some computer background or skills. Alternative concepts may vary in the degree to which other special knowledge--about learning and instruction and about knowledge-based systems technology--is needed by their users. The number of DEs for any problem domain who are potential users of the IKAS decreases as requirements for special knowledge increase. This factor can

make a system [illegible] of organizational constraints on the availability of [illegible] high-skilled OEs. An analysis of an IKAS' functionality provides information for estimating user knowledge requirements

USER TRACK [illegible]. For concepts in which [illegible] necessary to prepare OEs to [illegible] training, it [illegible] that training [illegible] to be [illegible] requirements [illegible] improvements in time-on-task [illegible] actively limit the applicability of the [illegible] by reducing [illegible] psychologically [illegible] that are useful for determining [illegible] the expected effectiveness [illegible]

ALTERNATIVES [illegible]

Our [illegible] described led us to the following conclusions [illegible] approach to automating the development of instructional [illegible] satisfy NAVTRAEQUIPCEN's objectives would be [illegible] The elicited knowledge would consist [illegible] other aspects of the task of instruction as [illegible] Such instructional systems--either a part-task [illegible] capabilities or an adaptive expert [illegible] This [illegible] work in the area of expert consultation systems, [illegible] would supplement or replace the [illegible] Unlike work [illegible] systems [illegible] knowledge-based instructional systems [illegible] (see Barr and Feigenbaum [1982, pp. [illegible] instructional systems).

Our IKAS [illegible] is supported by the analysis of the knowledge engineering task [illegible] (described [illegible] below). Prior work on aids for [illegible] (described [illegible] analyses [illegible] conclude that it is infeasible [illegible] arbitrary [illegible] problem [illegible] time [illegible] systems with the [illegible] to automate [illegible] about his domain and [illegible] knowledge of the [illegible] acted, prior knowledge [illegible] domain knowledge [illegible] These [illegible] knowledge [illegible] that imposed structure on its own domain knowledge [illegible] historically been useful only for KEs, not DEs.

[illegible] for a "general" IKAS is more modest in scope. [illegible] general IKAS system should be limited to elicitation of knowledge [illegible] is an instructional system with a particular architecture [illegible] set of instructional capabilities. Further, the [illegible] system could be used should be limited to those within a predefined "class" of problem domains. A class might subsume several specific [illegible] each of which uses a set of concepts and relations that [illegible] to those in the class but which may differ in their detailed content. For example, operating a radar may represent a class of tasks, the members of which are distinguished by the different

characteristics of and procedures required by different specific surface- or air-search radars.

Restriction of the scope of the IKAS in this fashion will permit the use of semantic and pragmatic knowledge of the problem class for effective management of knowledge acquisition interactions. Our concept therefore includes the use of class-generic knowledge by the IKAS to guide interactions with the user. The ability to use knowledge about a class of tasks to guide the acquisition of instructional knowledge about specific tasks in that class imposes two important requirements on the nature of the IKAS. First, the design of the instructional system must be stable, so the knowledge of structure and capabilities used by IKAS does not become obsolete and inappropriate. Second, the development of the IKAS for a class of tasks depends on at least a partial implementation of one application within that class. This implementation would assist the process of identification of class-generic knowledge and structure using general concepts induced from the specific task under study. The resulting IKAS would be capable of supporting the development of the identical instructional system for other members of the domain class.

The following section discusses three promising alternative realizations of the IKAS concept. The alternatives were selected by considering the criteria described above and assessing the cost and expected benefits of development of the concepts. We perceive tradeoffs among the criteria as applied to the alternatives that prevent us from identifying any one of them as best from a technical perspective. However, each of them appears to satisfy NAVTRAEQUIPCEN's stated objectives and entail a feasible set of system capabilities that extend the technology currently available to the developers of computer-based training systems.

ALTERNATIVE 1: AN AID FOR THE SPECIFICATION OF PERFORMANCE MODELS. Expert systems--knowledge-based systems designed to act as consultants for domain specialists--embody a competence model, the knowledge that ought to be used in problem-solving situations. Knowledge-based instructional systems (KBIS), if they are to implement performance diagnosis for trainee evaluation or selection of instructional interventions, also need performance models. These models contain representations of the knowledge learners may actually apply in situations. Performance models describe errorful or suboptimal behavior relative to the competence model. While expert systems need to generate actual competent problem-solving behavior, some KBISs have been able to perform performance diagnosis with descriptive, less complete competence and performance models, different from those employed in expert systems. As a consequence, constructing a KBIS can have substantially different requirements for knowledge acquisition than an expert consultation system (Brown, 1977; Clancey, 1981).

One alternative IKAS concept focuses on the acquisition of the knowledge necessary to construct performance models in a diagnosis module of a KBIS. This knowledge includes:

a. Descriptions of non-expert behavior (errors, parametric variations) in the domain

b. Characterizations of situations that evoke non-expert behavior from learners

c. Causal explanations of non-expert behaviors expressed as variations on competence model representations

Appendix [illegible] presents a hypothetical dialogue between a KE and a DE in Navy surface tactics that embodies these types of knowledge.

In this alternative, the knowledge constituting the competence model would be manually acquired using the standard method of KE interviews of DE. The KE would implement the KBIS architecture and encode the competence model prior to use of the IKAS. The KE would use that domain-specific knowledge in the competence model and the class-generic knowledge in both the competence and performance models developed in the manual implementation of the first KBIS for the class to elicit the domain-specific performance models from the DE.

This system [illegible] the following characteristics:

a. There appears to be [illegible] risk with respect to its feasibility. [illegible] successful efforts to develop and use performance models in KBIS have been for limited problem domains. Model development in these efforts has been a varied, ad hoc process compared even to competence model development in expert consultation systems. There are thus few precedents for manual knowledge acquisition on which to base an IKAS.[5] On the other hand, this alternative would have a relatively high payoff. Most of the potentially significant capabilities for KBISs, especially for use in simulator-based training, require automated performance diagnosis. In addition, development of an IKAS for the knowledge underlying performance diagnosis could contribute, as a side effect, to the maturing of KBIS technology. The relative risk and complexity of the proposed system and the lack of an existing system that might shortcut the required development of a first KBIS in the selected domain class imply that implementation of this alternative could be a significant research effort.

b. The [illegible] performance models is dependent on the competence model. The generality of this alternative would be limited to classes of domains for which the knowledge and formalism(s) used to represent their competence models would be appropriate to several specific domains in the class. For procedural types of knowledge, precedents suggest [illegible] appropriate formalisms do not generalize to loosely defined classes. Domain classes such as radar or sonar operation and troubleshooting of communication equipment seem to be candidate domains with an effort to degree of generality, but at this point such judgments are largely intuitive. For other domains (e.g., Navy platform

[5]For the most part, performance models have been articulated by system builders familiar with the domain rather than by domain experts.

defense tactics), it is not clear at what level of abstraction (ships within classes, classes within platform types, types within mission areas) generality can be achieved for this IKAS alternative.

c. It is not evident whether this system would reduce the DE's time-on-task during the development of a KBIS. The DE would be involved in both the manually-based development of the competence model and the IKAS-based development of the performance model. It is possible that the duration of both could be reduced over a fully manual approach because the IKAS will be able to use existing class-generic knowledge in both types of models. The KE's time-on-task would be reduced to the degree the IKAS replaces him in eliciting performance models from the DE. It is likely that these savings would be substantial because the knowledge embodied by performance models is typically many times greater than that of the corresponding competence model. However, total time savings would undoubtedly be offset to a degree by the KE's need to familiarize himself with the performance models. Although a key assumption of the general IKAS concept is that the KE will not undertake modifications of the KBIS architecture, there will certainly be some need for the KE and DE to refine and revise the knowledge acquired using the IKAS. To do so, the KE will need to spend time acquainting himself with that knowledge.

d. This IKAS concept will require some training for the DE so that he can use the system interface and understand the knowledge representation used in the KBIS. This background will be necessary to allow the DE to engage in meaningful dialogues and to provide useful information about potential variations from competent behavior and situational parameters that affect performance. An important advantage of this alternative is that such training can be incidentally embedded in the interactions between the KE and DE during development of the competence model. Those interactions will introduce the representation concepts to the DE. Further, if the KE works with the DE while using the IKAS interface to develop and test the competence model, then the DE should become sufficiently familiar with the IKAS that his later use of the system can be supported largely by reference manuals, with little or no need for formal tutorials. Thus, while this IKAS alternative requires background and skills for its use, we anticipate very little dedicated training will be needed for users to acquire them.

ALTERNATIVE 2: MODIFICATION OF AN OPPONENT SIMULATION FOR TACTICAL TRAINING. The second alternative for IKAS concept development addresses a training capability that does not depend directly on performance diagnosis and the performance models it requires. The capability involves providing the capability for the DE or instructor to define and sequence the set of problem situations to be presented to the learner. Providing this capability to alter the training system would allow the instructor to develop curricula, monitor and participate in training exercises, and control the course of instruction. The specific example of this capability we will consider here entails the modification of the behavior of an automated opponent in a tactical training simulation. (However, there are analogous capabilities in other domains--for example, the manner in which a piece of equipment will behave in a maintenance training simulator.)

The system concept assumes that the opponent simulation is based on a rule-based representation of the actions and procedures the opponent should execute in tactical situations. Further, it assumes that prior to use of the IKAS, the definitions of rules, high-level procedures, and primitive actions are sufficiently complete that the opponent simulation is operational. At this point during system development, its behavior will probably represent an initial formulation of a skilled opponent. Use of the IKAS would allow the DE to specify variations of rules and procedures governing the opponent's behavior. Such modifications would produce alternative, perhaps suboptimal, opponent behaviors--performance models for the opponent, as opposed to performance models of the learner as in Alternative 1. The variations might include both structural and parametric changes. Such modifications might be intended to achieve a number of goals--variation in skill levels, use or avoidance of particular tactics or systems, or the creation of particular tactical situations for the trainer to respond to. Such variants could be used for a number of pedagogical purposes, depending on the training goals of the instructor.

The domain of this system concept is similar to that pursued in previous research sponsored by NAVTRAEQUIPCEN (Alperovitch, 1982). That work, however, focused on the acquisition of an initial set of procedures and presented the user with a programming-like task requiring significant knowledge and skill. The approach advocated here uses the IKAS to aid the DE in specifying modifications of knowledge the system already contains. That knowledge, including class-generic knowledge about the characteristics of variations obtained in the manual development of a first system, can be used by the IKAS to provide greater system initiative in suggesting what variations might be generated and what sort of modifications might achieve them. In addition, because modifications are always made to an operational set of rules and procedures, the DE can develop them incrementally and exercise them in the KBIS through an interface to the IKAS. The DE can thus approach the task empirically without a fully developed abstract understanding of the opponent simulation.

This alternative has the following characteristics:

a. The relatively limited scope of the knowledge to be acquired by the IKAS in this alternative argues for its feasibility. The system concept reduces the complexity of the task of acquiring procedural knowledge by limiting IKAS activities to secondary elaborations, thereby avoiding initial formulation by open-ended specification. However, we do not perceive significant differences in feasibility and risk between this concept and Alternative 1. This alternative does have greater potential for increasing functionality of knowledge acquisition beyond that of manual methods. The ability to test modifications by executing the entire KBIS could increase the DE's productivity and decrease the need for KE involvement in post hoc knowledge-refinement. The generality of this alternative is limited to a class of tactical problems with a common semantics for situation and action descriptions: tactics problems involving physical threats and the movement of platforms.

b. This alternative may have a more rapid implementation time compared to Alternative 1 if it were coordinated with recent or ongoing

efforts to develop intelligent opponents for simulator-based training.[6] In this case, at least some of the effort required to implement a first KBIS in the domain class could be circumvented. Coordination would depend, from a technical standpoint, on the ability of the representations used in the existing efforts to support incremental elaboration. The implementation time advantage may be augmented by the relatively smaller scope of the knowledge to be acquired by the IKAS. We also perceive a greater requirement for completeness in specifying competence models (as in Alternative 1) than in specifying alternative models of opponent behavior vis a vis the instructional capabilities that might use those models within a KBIS.

c. The implications of this system concept for DE and KE time-on-task are similar to those for Alternative 1. DE time should be comparable to that in manual knowledge acquisition, while KE time should be reduced by the DE's use of the IKAS, with the net savings affected by the extent to which the KE must engage in compensating activities. The relative savings depend on what other capabilities are included in the KBIS (e.g., performance diagnosis and instructional interventions). Since this alternative for IKAS development does not affect acquisition of knowledge for those capabilities, the overall role of the KE in knowledge acquisition could remain significant.

d. This system would require DEs to have a significant understanding of the knowledge representation formalisms used in the KBIS. The predominantly procedural nature of the formalisms for implementing an opponent simulation (as opposed to descriptions of errors and situations in Alternative 1), would most likely require greater skill and knowledge of the IKAS users. As in Alternative 1, however, the prior person-to-person interaction between the KE and DE to develop the initial opponent model should provide the DE with the requisite knowledge for using the IKAS.

ALTERNATIVE 3: CONSTRUCTION OF KNOWLEDGE BASES FOR INSTRUCTION ON DOMAIN FACTS. The third alternative we describe emerged from interviews with system builders and DEs at the Navy Personnel Research & Development Center (NPRDC) (see Section V). In discussing our concept for a class-generic IKAS with them we learned that an NPRDC project had developed an experimental prototype of a generic KBIS (McCandless, 1981; Crawford and Hollan, 1983) and that attempts to have DEs implement knowledge bases for new problem domains had encountered difficulties. They believed that an IKAS for the system is necessary if its application to other domains is to become cost-effective.

The NPRDC system was first implemented for the surface warfare domain,

---

[6] Efforts we have identified include Contracts N61339-80-C-0079 (Naval Training Equipment Center) and N00014-82-C-0653 (Office of Naval Research). The Navy Personnel Research & Development Center has also proposed research on tactical decision making training that might encompass the use of opponent simulations.

where its objective was to teach the concepts and facts Tactical Action Officers (TAO) must memorize in order to perform threat identification, assessment, and counter-assignment. We will therefore refer to the system as the Computer-based Memorization System (CMS). The CMS was implemented with a highly modular architecture. A semantic network is used to represent the domain concepts and facts in a relational database. The database describes taxonomies of objects and object attributes and relates particular objects with their attributes. Instruction is provided by a set of seven modular "games" (e.g., "twenty questions", "concentration") each of which independently uses the semantic network to generate problems. Generation depends solely on the network structure, not the domain content.[7] Thus, the CMS architecture is actually generic to any problem domain for which concepts and facts can be described using its semantic network formalism and for which memorization of facts is an instructional objective. It has subsequently been applied at NPRDC to knowledge bases for the domains of EW, ECCM, and sonar analysis.

The current procedures for creating new knowledge bases for the CMS are not conducive to direct use by DEs. The knowledge base is created using an off-line editor and "compiled" for use in the CMS. The physical interface requires thorough knowledge of the programming system in which the CMS is implemented. The editor is separate from the CMS and places the knowledge-base development process solely under the user's initiative with no conceptual support--for example, by using existing knowledge bases and data about their development. All such conceptual support must be obtained externally by the user. The lack of conceptual support has manifested itself in the ability of one DE to master the physical interface and create a syntactically valid knowledge base, which nonetheless provided inadequate behavior in the CMS games because of the DE's inability to anticipate how the knowledge in the network would be organized and used to generate problems.

Thus, a third alternative for pursuing IKAS development is to build upon the existing CMS work. The objectives would be to provide conceptual support and a more accessible physical interface for the user. This alternative has the following characteristics:

a. Feasibility of this alternative is higher than for Alternatives 1 and 2. The existence of a generic KBIS already in use by DEs for knowledge base development is persuasive evidence for the tractability of the IKAS concept.

b. The generality of this alternative is somewhat different from the concept of generality we have introduced previously. In the preceding discussion, we used the concept of a class of domains, to which an IKAS and KBIS would apply. Each member of a class would have a significant overlap of semantics and pragmatics and utilize a common set of instructional methods. The CMS approach achieves generality by isolating a type of knowledge that is a part, possibly a very small part,

[7] Two exceptions are games specific to the TAO domain that use graphics.

of the knowledge in a very large number of problem domains whose other types of knowledge may be very different. It thus achieves generality across domains by greatly limiting scope within each domain. That is, the system would address only factual, conceptual knowledge, which is only a small part of what constitutes competence. One consequence of this approach to obtaining generality is that an IKAS for the CMS will be unable to use class-generic semantic and pragmatic knowledge for dialogue management. Instead, dialogue management will have to be based on the syntax of the network representation and on knowledge of how game capabilities access the knowledge base. Thus, this alternative appears to achieve feasibility and generality across domains at the expense of limiting the power of IKAS techniques available to the system and the completeness of the knowledge that can be acquired.

c. The implications for this alternative for DE time-on-task are again difficult to assess. Within the scope of applicability, KE time-on-task would be greatly reduced since, in contrast to Alternatives 1 and 2, there is no programmatic requirement for KE involvement in the knowledge acquisition process. We expect however that some ad hoc KE involvement for consultation and tuning would be required in most cases.

d. As an exploration of IKAS technology, this alternative would involve less time and resources than Alternatives 1 and 2. The generic KBIS exists and several problem domains have already been implemented using it. New effort could be focused on the IKAS architecture and implementation, unless some re-implementation of the CMS into a more tractable software environment was deemed necessary to support IKAS functionality. That the scope of application is limited to acquiring only declarative knowledge would also serve to control costs and implementation time for this concept.

e. Users of an IKAS for the CMS would need to have an understanding of the semantic network representation and its relationship to the mechanisms of the instructional games. This understanding would have to be developed by either explicit training or by training embedded in the IKAS. The latter training might utilize examples from previously implemented domains. The degree of initial training required would be reduced if the interface between the IKAS and CMS facilitated an empirical approach to knowledge-base development (as in Alternative 2). In such an approach, DEs could readily experiment with possible knowledge formulations using the CMS games. Because the knowledge involved is solely declarative, we anticipate that the physical interface to the IKAS would require computing skills similar to those required by a stand-alone word processor. In contrast, the specification of procedural knowledge, as suggested by Alternative 2, has some of the characteristics of programming and would potentially require more sophisticated computing skills.

SECTION V

IKAS DESIGN CONSTRAINTS

In this section, we consider further constraints on the design of an instructional knowledge acquisition system (IKAS). In the previous section, we enumerated several criteria for evaluating alternative IKAS concepts. These criteria reflected high-level technical and organizational concerns. The present discussion will consider issues that bear upon the design of user interfaces for IKAS systems. The sets of issues are not independent, nor is the design of an IKAS user interface independent of the purpose and features of the KBIS it serves. Our main objective for this section is to identify how requirements for easy usage of an IKAS influence the design of the system's user interface.

We first enumerate important dimensions of design variation in user interfaces, providing for each some background discussion of the current methods and techniques associated with that aspect of interface design. We then discuss the design requirements in the context of the general IKAS concept and its three alternative realizations presented in Section IV. The evidence supporting our conclusions about these requirements draws upon the literature on human interface design, but it comes primarily from statements elicited from a sample of Navy DEs, builders of state-of-the-art instructional technology, and developers of knowledge-based systems. We then present recommendations, which are based on assumptions about host system architecture and capabilities, for formulating a user interface design for a specific IKAS.

We do not present here a general review of the growing literature on human factors in the design of interactive computer systems. Such a review was beyond the resources and scope of the current project. In any case, our view of that literature is that detailed, substantive conclusions about effective interface design are specific to task applications, such as word processing and database query. The interactive computing tasks that have been studied are very different from the task of transferring new knowledge to a knowledge-based system. Hence, specific details of user interface design used in other interactive computing tasks are of limited use in designing the interface to an IKAS. More general principles of design of the type we will discuss are the common lore of system builders and are difficult to attribute to any one effort. They are also vague. For this reason, we refer to them as "issues" of design instead of as principles of design.

USER INTERFACE ISSUES AND METHODS

Our user interface issues, derived from discussions with Navy DEs and instructional system builders, are similar to others identified in the literature on interactive computer system design (see recent reports by Ramsey and his colleagues [Ramsey, Atwood, and Kirshbaum, 1978; Ramsey and Atwood, 1979] for a bibliography and review of this literature; see Martin [1976] and Simpson [1982] for similar classifications). A critical feature of the issues is their interdependence: particular interpretations of how an issue affects design constrain interpretations of other issues. A second critical feature is that the methods used to achieve a design feature associated with an issue

are generally not applicable to all interface designs. Instead, most designs involve hybrid methods.

MEDIA. The issues surrounding input/output media are perhaps the most obvious in user interface design. The CRT has become almost a universal output medium, but issues about display details--graphics, screen resolution, color output, multiple displays--remain. Other output media currently receiving attention include videodisc and computer-generated voice and sound. Input media include the typewriter-like keyboard built into most CRTs, devices for spatial designation and manipulation--touch panels, light pens, bitpads, "mice," and voice.

A major rationale for designs incorporating multiple media has been a general belief that distributing interaction functions among perceptually distinct channels is beneficial. There are more specific issues as well regarding input mode and the organization of interaction that are connected intimately with use of media. Input mode may scale from open to closed, where open denotes responsibility for the user to remember and structure "legal" inputs and closed denotes the system's responsibility to continually inform the user of "legal" inputs which he may select. The open mode of input is typically accomplished using artificial languages, which may approach natural language in their flexibility and complexity. Closed mode is exemplified by methods such as menus and templates.

Related to input mode is the organization of interaction. At one extreme, interactions can be organized linguistically. They occur on a dimension of time and use linguistic devices for reference to earlier events. At the other extreme, interactions can be organized spatially. They reference objects and actions by direct manipulation of the structure of visual models. The temporal connection between action and effect is discarded (at least in the context the interface provides to the user).

The linguistic organization of interaction can be combined with both open and closed input mode. For example, the system can provide its output as text and accept inputs as command language inputs, choices from a set of alternatives via a keyboard, or choices made by pointing to its text output. Many systems mix both input modes. Much recent interest in spatial organization has been motivated by the objective of making the closed input mode more efficient. However, spatial organization is also compatible with open input mode. Image editing systems, for example, accept typed inputs or sequences of movements and button signals from a mouse.

RESPONSIVENESS. The physical responsiveness of an interactive system is a common issue. Poor responsiveness is held responsible for both undermining motivation and directly disrupting the user's thought processes; either case reduces productivity. On the other hand, it is possible that response can be too fast for some tasks and some users when thoughtful behavior is required. The user should neither feel delayed nor pressured by the system's responsiveness. However, in practice too rapid response is not a serious concern of system designers.

Poor response has both delay and variability components. It is commonly thought that extreme variability is more disrupting than mere delay.

System response depends on the capacity and loading of the entire hardware-software environment. Ultimately this means that achieving a better responsiveness may entail eliminating possible features of the target system. Unfortunately it is usually difficult to predict system responsiveness prior to implementation. Thus, one method for addressing system responsiveness is to design and implement features in independent and separable modules to permit alternative system configurations.

Another method for addressing system delay is to signal the user about expected delays of response. This involves endowing the system with knowledge about itself so that it can indicate to the user when it is about to perform operations that involve considerable time.

The use of networks to gain access to systems on which the user software executes complicates the responsiveness issue. Networks introduce delay and variability that the original designers and implementors can only roughly anticipate. In addition, they reduce the utility of signalling methods, since it seems likely that no signalling is better than the incorrect and inconsistent signalling that a system operating across a network can provide.

FLEXIBILITY. Flexibility involves three characteristics of an interface:

a. Alternative input possibilities

b. Alternative output possibilities

c. Handling of errorful inputs

The first two of these are related to the needs and preferences of different users or of a single user accessing the system for different purposes. One common method for achieving increased flexibility is the use of modifiable profiles that allow settings of switches. These switches can be referenced by the user interface software to select alternative behaviors. Other common methods depend on the implemented physical media and input mode. One extreme form of flexibility allows both open or closed input specification when possible. Flexibility in open input is increased by designing a language with multiple devices for expressing the same message to the system. This includes redundant lexicon, alternative syntactic constructs, provision of defaults for elided input, and resolution of anaphoric reference. At a lower level, recognition of partial inputs also increases flexibility. For closed input, such as menus, flexibility is enhanced by use of multiple media--pointing devices, keyboard function keys-- as alternatives for selecting among the same responses.

A system in which errorful inputs cause a system to break--confronting the user with the interface of the underlying operating system and requiring a restart--is called brittle. Systems are made less brittle by methods such as prevention and trapping of events that cause operating system interrupts and by other "worst case" assumptions in input handling code. More advanced approaches add spelling correction (for open input mode) and other "do what I mean" capabilities for interpreting anomalous inputs. One rationale for the use of closed input modes is that they are a relatively simple technique for

making a system more robust.

CONTEXT. Preserving and providing user access to the context of the human-machine interaction is important for providing recovery from errors, especially those resulting from user inputs that are semantically meaningful but have undesirable consequences. The user needs to be able to recover from such errors by invoking an "undo" command. Preservation of context is also important for providing capabilities for users to easily interrupt their work session and resume it in the future. Methods for achieving this capability are typically linked to conventions for process and file manipulation specific to the host operating system.

User access to context is important in conceptually supporting the user in complicated activities with the system. The simplest type of context is feedback about the effects of user inputs. It plays a role both in learning to use a system and in preventing the propagation of semantic errors through a sequence of events that makes them difficult to undo. In a linguistically-oriented interface, feedback usually comprises descriptions of effects or paraphrases of the user's input. In a visual-spatial interface, feedback comprises perceptible changes in the visual model displayed to the user. In the latter case, the association between previous user inputs and their results is more difficult to represent, since the model typically has no natural means for expressing the order of effects on it. Thus, visual-spatial organization may make it more difficult for a user to remember or reconstruct what sequence of inputs led to a particular situation. On the other hand, that type of interface does make the net of all effects continually available to the user, whereas in a linguistic interface the capacity of the display medium limits the prior history that is immediately available. The traditional form of interaction history for linguistic interfaces is a hardcopy record. Newer methods dependent on high-speed, high-capacity displays make records of prior interactions available via interactive "browser" facilities.

Another issue involving context concerns the handling of multiple and subsidiary contexts. Systems for complex tasks may include different modes for different activities. The scope of a single activity may be too great to allow all relevant information to be displayed simultaneously. Under such circumstances, the user can lose track of the local context of his interaction. One approach to this problem is to always provide cues, either linguistic or perceptual, about the current context and its surrounding contexts. An additional method is to make transitions between contexts non-destructive by preserving them and providing mechanisms for returning to them. Such facilities can include user accessible information about suspended contexts and mechanisms for alerting the user when activities in other contexts are incomplete. This kind of active monitoring is dependent of the interface's use of knowledge about the task domain.

USER CONTROL. User control refers to the system's facilities for allowing the user latitude in the way he approaches his task. It overlaps with the issue of flexibility regarding control over how to enter inputs. More fundamentally, it includes questions about the degree to which the user can determine the order in which he performs different subtasks.

Interfaces that give the user extensive control of interactions require greater resources for context management. They generally require more initial learning time to be used effectively. However, they can enable more efficient use by experienced users. Interfaces that assume most of the initiative in interactions require some internal representation of an acceptable organization for the task. Thus, they are more difficult to implement for complex tasks. They can make heavy use of closed input modes and thus can enable easy use on simple tasks by naive users. The need for users to take initiative increases as the scope of a task increases. At the very least, they need the ability to return to earlier interactions to change their inputs or review the events without losing the intervening interactions. One method for dealing with the user control issue is to provide capabilities for both full user control and for "hand-holding" by the user interface, with the ability to switch between modes as desired. This approach requires maximal support for context management and more intelligence in controlling system-determined interactions than can be provided by a rigid script.

USER KNOWLEDGE REQUIREMENTS. Any system design must consider what knowledge users must have to use a system and how they will obtain it. It is obvious that the user should not need to learn more than necessary to perform his task in an effective way. Layered designs can help minimize user knowledge requirements. For example, insulating the user interface functionality from the underlying system structure helps assure that the user need not know about the host operating system. However, other knowledge of the system is necessary to promote effective interactions, just as a conversation between two people requires certain shared knowledge and assumptions. At the very least, the user must have knowledge about the models of the task domain that the system incorporates so that he can use them or map his own models, if any, onto them; without such knowledge, the user cannot predict the effects of his interactions.

The user can also perform more effectively if he has an understanding of the interaction schemas the system employs. This can include various types of knowledge: the plan underlying system-controlled interactions, the feedback he can expect in different situations, the syntax of open input mode languages, and the conventions for accessing and managing context information. Most of these, of course, depend on how other issues regarding the user interface have been resolved, so in effect the other issues all have implications for user knowledge requirements.

Approaches to satisfying user knowledge requirements may be categorized as external or internal to the target system. External approaches may be based on either selection or training. Selection of users whose aptitudes and prior experience better equip them to interact with a system is possible in limited cases where the system's purpose does not require universal accessibility by a population. Thus, for example, while it is necessary for every teller in a bank to access a system for recording transactions with customers, it may only be necessary for a subset of them to access a system for establishing new customer accounts. Training based on off-line materials and presented by human specialists is a common method for satisfying user knowledge requirements. This external training is typically not adapted to user needs: it is unnecessarily expensive for some trainees and insufficient for others.

Internal approaches to satisfying user knowledge requirements attempt to increase accessibility by making presentation of the knowledge concurrent with use of the system and adaptive to user needs. These approaches combine several facilities in the system's user interface. Usually, default control of interactions is assigned to the system; that is, the system does maximal "hand-holding" for new users. The interface also may provide integrated on-line documentation and assistance. In any situation, the user can follow a simple protocol to obtain a description of what is happening and what is expected, without destroying context of the interaction. When the user makes an entry error, corrective feedback and tutorial descriptions are presented or available. In effect, individualized training is delivered as needed and on request while the user is using the system. Another feature seen in internal approaches is layered capabilities. Instead of opening the full functionality of the system to the user, complex functionality is made accessible over time either under the control of the user or an external monitor. Thus, users need not satisfy all knowledge requirements before they can effectively use a system for some subset of activities.

DISCUSSION. The major conflict in resolving most issues about user interface design is in balancing the desire for access by new and casual users against the desire for efficient use by knowledgeable, long-term users. These different classes of users are generally served best by different input modes (closed vs. open), balance of system vs. user initiative, and degree of control over preservation and access to context. The difficulty in addressing both sets of needs in a single system lies primarily in the cost of engineering an array of alternative mechanisms and the difficulty of designing techniques for smooth transition among them. Engineering costs include both the software design and the hardware resources. Increasing the features of an interface lowers the quality of its responsiveness, which can make a system unacceptable to both classes of users. Maintaining responsiveness as the number of features increases requires enhancing the capacity and speed of the hardware system. If hardware resources are fixed, then the interface design must involve tradeoffs in functionality.

INPUTS FROM SYSTEM BUILDERS AND POTENTIAL USERS

As required in the contract statement of work, we interviewed several potential DE-users of an IKAS in order to gather data--expressions of their "needs and expectations"--that might be relevant to resolving issues in design of the system's user interface. We also obtained from the potential users information that bears on the question of whether they can articulate the knowledge about their task domain that the Alternative 1 IKAS would be designed to elicit.

In addition to potential users, we interacted with system builders who have worked with DEs to build training systems. We discussed their opinions about the feasibility of constructing an IKAS that could interact directly with DE-users. We also interacted with system builders with experience in building expert systems for knowledge acquisition. The following sections report the results of this data collection effort.

MODEL

Selection of respondents. We sought Navy DEs who had had some experience in development or use of training technology. We wished to establish with our interviewees a mutual understanding and rapport concerning our objectives. Therefore, we decided that communication would be enhanced by (a) our knowledge of the respondent's field of expertise, and (b) respondent's familiarity with computers and computer-based instruction. Practical limitations included our ability to identify and arrange access to active Navy personnel with demanding work schedules.

Our DE sample comprised four individuals contacted either through our colleagues in the Navy R&D community or through our activities in other Navy-sponsored R&D projects. Three are DEs in surface warfare tactics, and one is a DE in sonar operation and interpretation. These individuals were assigned to positions at the Navy Personnel R&D Center (NPRDC) and the Fleet Combat Training Center (FCTC) in San Diego, California, and had authority to interact with Navy R&D contractors at their own discretion as part of their job. They were either actively involved in training technology development or in training.

One group of system builders comprised six civilian Navy employees at NPRDC. These individuals are engaged in R&D efforts to design and implement prototype training systems for a variety of applications. Some of these individuals had been involved in the same system building efforts in which our DE respondents had participated.

We also interacted with two system builders at FCTC. These individuals were Navy officers who were building a system to be used in constructing embedded training for the Navy Tactical Data System (NTDS)--in some sense, an IKAS. Their particular concern was in user interface design to support access to the system by untrained, low-skilled users (i.e., not DEs) whose task would be to enter existing hardcopy specifications of curricula into the system.

We interviewed one other Navy system builder at the Naval Oceans System Command (NOSC) in San Diego. This individual had been identified by our respondents at NPRDC and NOSC as being currently involved in building a knowledge acquisition system for a Navy expert system.

At various points during the project, we discussed with colleagues from the Stanford University AI community the feasibility of automating knowledge acquisition from DEs and our particular concepts for an IKAS. These individuals all have experience as knowledge engineers.

Interview method. Interviews were arranged and conducted by Keith Wescourt of the project staff. Prior to the scheduled interviews, copies of the project's objectives from the contract Statement of Work were mailed to the DEs. Each confirmed that he had read this material prior to the interviews.

The interviews were held at the respondents work places over a 2-day period. By their own choice and our agreement, the DEs at NPRDC and at FCTC were interviewed in single group meetings. The FCTC meeting also included the two Navy officers who were developing the NTDS curriculum development system.

The DE interviews were conducted in an informal style. Our staff member introduced topics from a prepared agenda when and if they seemed appropriate

based on the respondents' apparent knowledge and opinions expressed during the interview. Since the topics were for the most part outside the day-to-day concerns of the DEs, they were encouraged to respond to inquiries based on their past, perhaps idiosyncratic, experiences. For example, they were encouraged to evaluate user interface input modes by reference to particular computer systems they had used. The discussion with each group lasted 2 to 3 hours. Both were tape recorded with the permission of the participants.

The interviews with Navy system builders were conducted as informal collegial discussions. The 1- to 2-hour interviews at NPRDC were also recorded. Interviews with other Navy system builders were arranged opportunistically during the visit to NPRDC. These interviews were shorter (10 to 20 minutes) and were not tape recorded.

Discussions with artificial intelligence researchers with expert systems building experience in the civilian world occurred informally at several times in the project. We will not describe these discussions in detail, but we will indicate when the opinions expressed during these discussions either reinforce or contradict those obtained from the Navy system builders.

Interview agendas. The agenda for discussions with the Navy system builders included the following topics:

a. What is the most difficult aspect of knowledge engineering?

b. How feasible is automation for knowledge acquisition? To what extent does feasibility depend on already having task and domain specific knowledge embodied in the automated knowledge acquisition system?

c. What user interface features does a computer system require to support direct use by Navy DEs?

These topics were discussed in the context of an initial description of our concept for an IKAS generic to a class of tasks. The respondents introduced other topics which they thought were relevant to our system concept, including the possibility of applying the concept to the enhancement of an existing instructional system prototype (Alternative 3, described in Section IV).

The agenda for the discussions with the DEs included a variety of specific questions for each of the following general topics:

a. What is your experience in using computer systems? (How long? For what purpose? What physical systems? What conditions of accessibility? What training?)

b. What makes a computer system "friendly" to its users? (What are the strengths and weaknesses of systems you've used? What general problems do you perceive? What changes would remedy them? What preparation should be required before first using a system?)

c. How do you see the Navy with respect to the general trend for increasing computerization of society? (How do you place yourself with

respect to that trend? With respect to the rest of the Navy? Are there important differences within the Navy by age or job area?)

d. How useful have computers been for operational and training functions for your job area? (Are there other opportunities? What inputs from DEs like yourself would be needed to develop the opportunities effectively?)

e. What experience have you had in interacting with groups developing systems for operations or training? (How did the involvement occur? Were there specific rewarding and dissatisfying aspects? How much background knowledge had to be exchanged before you felt progress occurred? Do you have any thoughts on how computers might have aided the system development process? What are the advantages and disadvantages of becoming involved in such projects?)

f. What is the nature of knowledge in your field of expertise? (How much of what is required for performance can be learned from formal coursework? How important are exercise and OJT? Is it hard to describe the knowledge derived outside formal coursework? How do you judge your ability to evaluate another's performance in your job area? When you see someone make a performance error in your job area, is it easy to see its causes? How long would it take to describe your knowledge of how to do your job? How would you help yourself remember and organize the knowledge you would describe? Do you think there are computer techniques that could allow you to communicate your description directly to a system?)

Time constraints and differences in the DE's backgrounds made it impossible or inappropriate to discuss with each all of these topics. For those topics covered, the DEs were asked for their belief about how typical their responses might be of others in their specialty and of how they might expect responses from individuals in other specialties to differ. We adopted this tactic in recognition that although our approach to selecting and interviewing DEs could obtain candid and detailed data, these data might be of limited generality. By asking explicitly about generality we hoped to obtain some qualitative characterization that would be of value for preliminary system design of an IKAS intended for a broader user population or other task domains. In any case, the current posting of these DEs to R&D and training billets suggests that they are representative of the individuals who would be prime candidates for the role of user of an IKAS. Therefore, it is reasonable to assume that the DES' remarks represent the expected norm for other, potential users in their job specialty, although the generality of their remarks to other task domains and domain experts may be more questionable.

INTERVIEW RESULTS: SYSTEM BUILDERS.

On the feasibility of automating knowledge acquisition. Every system builder we spoke with agreed with our analysis that replacing the KE with a generic knowledge acquisition system is far beyond the state-of-the-art. Their reasons included those we have stated previously:

a. The extensive knowledge and interpretive skill required initially to communicate with DEs and to organize interactions effectively

b. The dependence of knowledge acquisition on representation formalisms used in the host system

c. The dependence of choice of formalisms on domain features and on specific capabilities to be realized by the knowledge-based system, neither of which is known initially.

The specific features of the interaction between the KE and DE that were perceived as difficult to automate included:

a. The KE's role as "filterer and structurer", not just a "bridge" for transferring the DEs descriptions

b. Determination of the consensus within the domain for a specific DE's knowledge

c. The KE's use of evolving semantic and pragmatic knowledge to control the topics and granularity of interactions.

The last of these features bears on the issues of system initiative and flexibility in user interface design. It questions whether a system that takes active initiative in a knowledge acquisition system can ask the "right" questions--those effective for obtaining knowledge while motivating the DE--using dialog management rules based only on the syntax of knowledge formalisms.

There were varying degrees of concurrence that an automated knowledge acquisition system generic to a class of tasks is feasible at this time. This concept was perceived as embodying a moderate level of risk, but one appropriate for a research effort. That is, even if the effort failed the process would have provided valuable knowledge to the community of system builders. One of the respondents felt that the major problem would involve operationalizing the notion of a class of tasks for use by an automated knowledge acquisition system. He wondered whether the requisite class-generic knowledge could be derived from one exemplar of the class or, if not, how many exemplars must be studied. Another concern involved the uncertain difficulty of using that knowledge as a KE does in dialogue management. Respondents who had developed a "generic" instructional system (CMS--see Alternative 3, Section IV) believed that that system instantiated the concept of a class of tasks. They believed that the difficult problem in implementing a knowledge acquisition system for such a system lies mainly in creating a user interface accessible to DEs.

On DE knowledge. Although it was not included among topics to be discussed with the system builders, the question of the nature of DEs' knowledge and its implications for our IKAS concept emerged as a serious issue. One aspect of that issue is the lack of consensus in expert task knowledge across DEs. The respondents at NPRDC who had worked closely with Navy DEs to build prototype training systems for several tasks (TAO threat classification, EW signal

classification, navigation) emphasized that DEs vary significantly in their knowledge and high-level approach to their tasks.[3] Such differences can exist even in the absence of variations in the level of task performance; that is, two experts may perform well with different knowledge and different approaches to the task. These variations reflect the lack of a true "expert model" and of good criteria for performance evaluation for these "navy tasks. The system builders agreed that their knowledge engineering activities required considerable interaction with DEs, exploration, and innovation to define a competence model for a task--unlike the knowledge engineering required for the well-known [illegible] consultant systems. One respondent suggested that consensus may be easier to obtain for "operator tasks" such as steam propulsion or radar operation. This comment was based on knowledge of an NPRDC project to develop a knowledge-based training system for steam propulsion plant operation.

The respondents at sites who had worked with DEs also indicated that few, if any, DEs could articulate rules for effective instruction of their specialty. As one respondent characterized it, DEs have "forgotten what it is like to be a novice." This opinion bears on the scope of an IKAS. Prototype systems based on knowledge about how to teach [illegible] of troubleshooting have been ineffective and unattractive to trainees who have used them. When these systems were redesigned by systems builders with training expertise, they were evaluated more highly and, in addition, the DEs acknowledged that the training methods were superior to those they had proposed. Several respondents offered an opinion that bears directly on the Alternative I concept of an IKAS for performance models of the types and causes of trainee errors. They suggested that while DEs are able to describe the types of errors a trainee might commit when performing a procedure, they do not have rich knowledge about the underlying causes of the errors (i.e., the trainees' "conceptual bugs") or about instructional interventions that are effective for eliminating errors.

On making systems accessible to DEs. The system builders perceive a range of skills in and attitudes toward computer use by DEs. From their point of view, the existence of a few DEs within a specialty with some computer background would be sufficient to justify a system with which DEs could interact directly. These few individuals are the ones who seem to participate currently in system acquisition efforts. A concern was voiced by arbitrarily selecting DEs with a specialty whose [illegible] activity. Manual knowledge engineering efforts with individuals who rank low in this manner have not been productive, and there is no reason to expect efforts assisted by automation would be more successful.

The respondents had a number of opinions about interface design. These were based on their efforts to build systems for use by DEs on tasks such as curriculum design and instructional system management. Again, the DEs participating in these efforts had good skills for non-computer professionals, were volunteers, and hence presumably motivated. Thus, the respondents'

---

[3]As we shall discuss, the DEs agreed totally with this assessment of their consensus.

opinions are based on a "best-case" appraisal of their experience. However, they also indicated how they expect the interface would have to differ for less experienced, but equally motivated, users of a system. Their suggestions are consonant with the conventional wisdom about interface design. In addition, a few specific suggestions oriented to the IKAS concept were proposed.

The inputs concerning interface design include those offered by the two officers at FCTC who were developing an operational curriculum entry system for NTDS training materials. Their users were untrained enlisted personnel and their ideas about interfaces were oriented toward state-of-the-art techniques for supporting such users.

The respondents at NPRDC have found that the skilled and motivated DEs who they have worked with have learned without difficulty to access systems through the same user interfaces used by the system builders themselves. This includes learning operating system commands to invoke the application programs, interacting with application programs that require user initiative, and using poorly documented and inflexible programs (typical characteristics of incomplete research prototypes). One comment about the experience of a DE using CMS is revealing, however. The DE learned to use the system, but had difficulty in accomplishing his objective of constructing a data base for a new application of the system. This emphasizes the distinction between how the mechanical aspects of an interface influence accessibility and how they influence the effectiveness of access. In the latter case, the user interface can be more or less effective at supporting the user's conceptual problem solving. Similarly, civilian system builders also indicated that since encoding knowledge into a system is difficult even for a KE, a major problem for automated knowledge acquisition is providing conceptual support, not the mechanics of system access.

Several comments were made about features of the user interface design that would in general be associated with the implementation of conceptual support. One respondent suggested that DEs are both more effective and comfortable with the elicitation of procedural knowledge while solving actual problems. Another respondent suggested two approaches. One was to support specification of new knowledge (new inputs) by indicating how it is different from existing, analogous knowledge, instead of by composition of primitives from scratch. The second was to support completion of a knowledge base by generating structural (syntactic) variations automatically for the user to filter, instead of by relying on the user to remember or formulate the variations on his own initiative. The respondents did not identify these features with the need for any particular characteristics of the physical interface.

With respect to the question of user interface requirements for DEs with less skill, experience, and motivation, the respondents' replies were quite general. One of them believed that for such users, the interface should take maximal initiative. She held that the need for structure outweighed the possible adverse implications for flexibility, even if the resulting interaction was so rigid and pedantic as to offend some users. Another respondent pointed out that interfaces requiring typing severely reduce the effectiveness for users with less experience. His comment seemed to suggest that linguistic organization of interaction, as well as the input mode and

medium, was less effective than the alternatives for such users.

The FCTC respondents reported positive results for a system (LTRAN, "Lesson TRANslator") used by inexperienced users. The features of the LTRAN interface include maximal use of menus responded to by function keys, spatial visual organization of the interaction, a pointing device for graphics interaction, "intelligent" on-line "help" facilities, and a user-invoked "undo" function for correcting mistakes. The spatial-visual organization is enabled by the nature of the task-- the construction of training lessons for delivery on the NTDS graphic display. The user of LTRAN continually sees the lesson as it will appear. The FCTC respondents feel strongly that the user interface features in LTRAN are important for supporting inexperienced users of other systems as well. In particular, the use of graphics displays and interactions increases accessibility. However, they also indicated that spatial-visual organization would probably be less effective for tasks in which there is no existing, natural use of graphics. They believed that requirements to learn an artificial visual model could increase the difficulty of using a system.

To summarize our discussions with system builders, we found:

a. They believe that any effort to automate knowledge acquisition must limit the intended generality of the system.

b. They emphasize the difficulties in providing conceptual support as the major problem in implementing a limited knowledge acquisition system for direct use by DEs.

c. They deemphasize the importance of the physical features of the interface to be used by DFs, because they believe that in general there are some DEs in any specialty with the skills and experience to use any functionally complete interface.

INTERVIEW RESULTS: DOMAIN EXPERTS.

On computing background and attitudes. All of the DE respondents had a working knowledge of interactive computer systems, a realistic attitude about the capabilities and uses of computers, and a favorable attitude about the introduction of new computer technology into training and operational environments for their specialty. One of the FCTC respondents had had formal coursework in computers at the Naval Academy and in an MBA program at a civilian university. The work at the Naval Academy included BASIC programming and use of a CAD system in a ship design class. He had not done any programming since leaving the Academy some 4 years earlier, however. Since that time, he had used a variety of embedded computer systems within the NTDS and other tactical warfare systems. At FCTC, he had a major responsibility for using NAVTAG, a microcomputer-based training system, in a supervisory role. He defined new exercise scenarios, played the role of game director in exercises, and managed trainee data in the system.

The other, more senior, FCTC respondent had no formal exposure to computing. However, his position required him to use various computer-based

training and administrative systems at FCTC. He also had experience using personal computer systems outside his work.

The NPRDC respondents had considerable computer skill. One had an extensive formal background from work in Masters and Ph.D. programs at the Naval Postgraduate School, which included 4 years of programming simulations in FORTRAN. At NPRDC, he was the DE member of a team developing tactics training systems. Part of that work included an independent research effort in which he was developing on a micro-computer in PASCAL an interactive, graphics-based, training simulator. In addition, he accessed NPRDC's VAX/UNIX timesharing system on a daily basis for computer mail and text-processing activities.

The second NPRDC respondent was a senior non-commissioned officer with self-trained computing skills. He had taken a COBOL programming course and had taught himself PASCAL programming at NPRDC. He reported that he was motivated to find ways to develop and use his skills in his job at NPRDC, even though this was not strictly required. He regularly used computers to interact with programmers implementing systems in groups in which he participated. He also was the DE who had worked on adding a new database for his specialty (sonar interpretation) to an existing training system prototype (CMS). Thus, he had actually been engaged in an "automated knowledge acquisition" effort.

With regard to the generality of experience and attitudes we encountered in our sample of DEs, the respondents at NPRDC recognized their relative depth of experience. The tactics expert believed his range of experience, particularly in programming, is uncommon for other officers in his specialty and exceeded only by officers whose specialty is computer systems. He noted that the other DE's knowledge and motivation was particularly rare for non-commissioned officers with real expertise in some operational specialty. That DE concurred by pointing out that other noncoms at NPRDC, including those in his specialty, had a very different approach to their involvement in projects developing computer-based systems.

The DEs at FCTC see their experience and attitudes as more common, though far from universal. They stated that among Navy officers in all specialties, one can expect a functional computer background from all who attended the Naval Academy, some of those who attended the Naval Postgraduate School, many who take courses at colleges and universities on their own initiative, and those whose specialty includes operational equipment that contain computers. With regard to the last category, one of the respondents noted however that the functionality of most embedded systems is less complex than that of the typical stand-alone word processor. One of their comments was that FCTC may be somewhat unusual in the extent of its use of computers. This may contribute to initiative of Navy personnel in developing computer skills. They were unsure, for example, whether the widespread ownership and use of home computers--five or six of the staff in the TAO training group are in this category--is typical of other Navy personnel. One factor they cited that affects such initiative is the amount of time required by assigned duties. It seems reasonable that the opportunities and motivation for self-improvement of computer skills generalize to other Navy training and development centers where DEs come into contact frequently with internal and contractor computer projects. DEs assigned to sea duties and to shore-based operations facilities

are less likely to have both the time and the stimulation to extend their experience with computers. In any case, the FCTC DEs believed it would be rare to find a current officer who would resist learning to use a new system if he were told its advantages. They suggested that resistance to computer technology occurs only at senior management levels. In attitudes toward computers then, the Navy appears to be at least as, if not more, progressive than other organizations.

On the features of user interfaces. The DEs had a variety of opinions about user interfaces. Many of these were vague and general; others were specific and described in terms of particular systems they had used. As we noted, we used our perceptions of these opinions to classify the set of "issues" about interfaces we described earlier in this section. Somewhat surprisingly, the DEs had more sophisticated, detailed perspectives on the physical features of user interfaces than the system builders we interviewed. Some of their comments at least partially contradict those given by system builders. One problem the DEs consistently had during our discussion was keeping separate the criteria they would apply to systems used by others (e.g., trainees using training systems) from those they would apply to a system they would use (e.g., the instructor's interface to a training system or the interface to an IKAS). Our discussion will be limited to presenting comments that bear on the latter type of user.

Much of the discussion with the FCTC respondents addressed the NAVTAG system's interface, with which both they and the interviewer were familiar. NAVTAG makes heavy use of sequential and nested menus. The respondents noted that while this feature was useful very early in one's use of the system, it quickly became tedious for both trainees and instructors. One aspect of the problem was responsiveness: the hardware/software was slow in displaying each new menu.[9] Comments about other systems indicated that responsiveness was a major issue for one of the DEs. He suggested that while some users could tolerate poor responsiveness, other users would elect not to accept and use a system with poor response time.

A second problem with NAVTAG's use of menus was the inability to immediately take a particular action when they knew exactly what they wanted to do. That is, the system is inflexible. They said they would prefer a command language interface, with some minimal prompting, that would give them more initiative. In addition, since they believe that DEs have minimal typing skills, the command language should be terse and the interface should be flexible in its response to spelling or format errors. One respondent suggested that the menu-based, scripted elicitation of responses under system control was not objectionable if the task was to enter a number of connected inputs (as in filling out a template), but that the user needed more control when choosing the topic of interaction and when modifying a single feature specified in a prior interaction.

Both DEs agreed that a rich, quasi-English command language was

[9]The next version of NAVTAG, under development on newer, more powerful hardware, could improve responsiveness.

unnecessary for Navy personnel. They routinely use arbitrary, schematic information-coding formats with rigid syntax in a variety of communication contexts--they even have to write computer-parsable messages. They do not object to learning such languages and favor them for their efficiency and minimal typing requirements over more verbose forms of expression.

Other responses about the use of menus bear upon the issue of context and feedback. One comment addressed both the preservation and display of context. The DE criticized the fact that the menus occupied so much of the display area and that each menu erased the preceding interaction. He thought it important for the user to be able to review and perhaps change responses in prior interactions. Another comment cited as a bad feature "being stuck in a loop," a situation in which the system rejects a response but neither tells the user what is wrong with it nor allows him to escape to on-line help or documentation without destroying the context of his work.

The FCTC DEs were receptive but not enthusiastic about the use of graphics in interfaces. They saw no particular advantage to graphics, especially the use of icons, if the images were not already familiar to the user. One DE thought such use of graphics is eye-catching but probably no more effective than alternative approaches requiring less expensive hardware and software. In addition, use of graphics does not preclude errors since the user must still remember sequences of actions. The other DE was also concerned about cost, suggesting that graphics are probably not cost-effective in infrequently used systems. Both DEs became more receptive to possible uses of graphics when the interviewer presented a hypothetical task of specifying a decision network for sequencing training exercises and illustrated (using a pencil-and-paper protocol) how graphics I/O techniques might be used. They thought the graphic display of the network as a graph was conceptually useful. However, they did not feel that graphic input (via a mouse or other pointing device) would necessarily be better for the user than the use of menus, especially if the menus and graph display could be simultaneously displayed. The medium and organization of interaction was important to them only to the extent that it could improve the amount of context available to the user.

The NPRDC DEs had a different perspective on user interface issues. Their comments tended to reflect the current common wisdom about interfaces. They also contradicted the system builders, who had deemphasized the need for "user-friendly" interfaces for experienced and motivated DEs.

The NPRDC DEs believed that while they had the skills to use arbitrarily complex interfaces, a good system design would not require extensive learning. The DE who had worked on developing a database for CMS stated that he should not have had to learn the operating system interface and PASCAL language to attempt that task. He believed that characteristics of the physical interface increased the difficulty of the conceptual part of his task. Beyond the physical interface's shortcomings, he suggested that the interface should have provided conceptual support for understanding the data base structure and its relationship to the instructional games. He felt that an application system for DE-instructor users should have the same interface features usually found in training systems themselves: turn-key access to the application, built in training, initial system guidance and prompts, function keys, and descriptions and manipulations oriented toward the organization and constructs with which the user conceives his task. Several other comments reflected his strong

belief in tailoring system capabilities for ease of use. He suggested that external documentation must avoid "burying" critical details about critical basic actions, that system designs should trade capabilities off to improve learnability and effectiveness, and that the need for multi-media and multi-mode user interfaces increases as the capabilities of the system become more varied.

This DE also provided the only comments about graphics from the NPRDC respondents. He believed graphics are only important when they reflect solid analogies with the user's existing understanding of the task. With respect to the interviewer's presentation of the hypothetical decision network elaboration task, he commented that direct graphics manipulation via a pointing device would be the best input mode for the task because of the ability to manipulate structures directly and ability to immediately view results of the manipulation.

The other DE had similar views. He was familiar with state-of-the-art "work stations" with significant built-in user interface facilities. He believed that hardware has been the main limiting factor on interface quality and that work station technology was overcoming these limits. He cited, for example, the original NAVTAG's inability to simultaneously display a geographical plot and tabular status display as a hardware limitation on the interface that adversely affected the system's usability. He stated that ease of initial learning, either off-line or by experimenting with the system was critical--he thought one hour of introduction prior to serious use was a safe maximum. He felt two design features would insure the utility of these learning sessions: (1) the use of system initiative and closed input modes (e.g., prompting, menus, function keys, and pointing devices), and (2) transparency (matching the structure and granularity of interface interactions to the user's conception of the task).

On DE knowledge and IKAS feasibility. We obtained only limited data from the four DEs that bear on IKAS feasibility. The role of a DE in building a knowledge-based system was unfamiliar to the FCTC DEs, so they were able to contribute little to this analysis. The NPRDC DEs were familiar with that role and had somewhat more to contribute. The DEs emphasized the lack of consensus about competent performance in their specialties (tactics and acoustic analysis). One NPRDC DE described the variability in experts' tactical decision-making approaches and tolerance for that variability as the result, at least in part, of the tasks characteristics: uncertain data, a large search space for problem solving, and the heuristic nature of available procedures. Since no one is ever always right, alternative approaches are acceptable. One of the FCTC DEs attributed expert performance variability to the competitive nature of the tactics task which introduces non-determinism and prevents a straightforward assessment of action consequences.

The second NPRDC DE commented on the tacit nature of expert knowledge in his specialty. His experience is that experts' rationales for their behavior do not coincide with the behavior they exhibit. To induce experts to give complete, consistent rationales, it is necessary to confront them with contradictions between what they do and what they say they do.

The DEs confirmed the opinion of the system builders that they do not have a good understanding of why trainees make specific errors of performance.

This is in part due to their poor understanding and consensus about competence. The DEs at both FCTC and NPRDC asserted that in actual training, verbal interaction is ordinarily required for an instructor to form a hypothesis about a trainee's underlying knowledge deficiency. One NPRDC DE believes that most fleet-based DEs do not have the verbal skills for such interactions and, further, that rank differences inhibit fruitful interaction during training. He did suggest that performance diagnosis from behavioral data might be easier in other specialties where there was less dependence on uncertain situational data and on data interpretations in determining which procedures to execute.

The other NPRDC DE commented that most DEs have little opportunity to develop detailed knowledge about trainee errors because training includes little "over the shoulder" monitoring. One of the FCTC DEs did believe he had developed diagnostic knowledge for trainee errors from his experiences at that facility. However, he acknowledged that the nature of the knowledge was such that performance data from many exercises would be necessary to converge on diagnostic hypotheses about a trainee's performance.

Only the DEs at NPRDC offered comments on our concept for an IKAS. One noted that our concept of a class generic IKAS was consistent with his experience in sonar interpretation, where there are about five different specialties that all do essentially the same problem-solving task using different equipment and having different coordination responsibilities. The other thought that automated knowledge acquisition was feasible, but that any effort to develop training systems would probably be served best by a combination of manual and automated methods. He also thought that an incremental approach to developing the technology would be critical for its success.

One mitigating concern was that if development of knowledge-based systems became more common, there might be a dearth of DEs with appropriate background and motivation to assist development efforts. In particular, unmotivated or inexperienced DEs might fail to contribute to the development of a KBIS regardless of whether knowledge acquisition was manual or automatic or of what user interface the automated system had. The other DE shared this concern, stating that common management practice of assigning personnel rather than soliciting volunteers could negate the effectiveness of any large-scale programs of knowledge-based systems development. He believed that suitable DEs would be indifferent to whether they worked with a human KE or an automated system on a system-building project. That is, that they would find working with an intelligent automated system acceptable.

DISCUSSION AND RECOMMENDATIONS

DISCUSSION OF INTERVIEW RESULTS. The limited and informal survey of system builders and DEs can only be generalized with care. All respondents were members of two small Navy communities (other than the Stanford system builders with whom we interacted informally and the Navy scientist we interviewed by telephone). These communities are geographically proximal and frequently interact with one another. The DEs represented only two job specialties that are somewhat related in function. On the other hand, they represent a range of roles for individuals with those specialties and they are the types of

individuals who would most likely participate in R&D on knowledge acquisition. Further, the DEs and system builders had interacted on some projects and had shared other experiences involving the development and use of particular computer systems. Thus, the comments we received reflect perspectives of system builders and DEs who might be using an IKAS together today if one existed.

The responses that strike us as most important are those indicating that the effectiveness of automated knowledge acquisition depends first on selecting appropriate DE users, regardless of the system's user interface. Appropriateness seems to refer to the DEs motivation and basic understanding of computers. Since we were told that most DEs of officer rank have the latter, it seems that motivation is the key factor. By itself, a user interface--no matter how user-friendly--will not guarantee that an arbitrary DE will become a productive member of a knowledge-based system development effort.

Beyond this point, we observed considerable divergence of opinion. The system builders believe that no particular physical user interface characteristics are required for a motivated DE collaborating in a development effort. Their experience indicates that such DEs are willing and able to learn to use the same physical user interfaces they themselves use. They do believe that any system must provide conceptual support for the user. Thus, the "what" of human-computer interaction is important, but the "how" is not. The DEs expect such conceptual support, but they also want a friendly physical interface to the system. Although they can and do learn to use systems with idiosyncratic and complicated user interfaces, they find this to be an imposition and an impediment to achieving their goals in using those systems.

The DEs expressed no consensus on what features they desire in a user-friendly interface. Like the system builders, they recognized that tradeoffs exist and that interface design depends on objectives and functional capabilities of the system. The DEs were largely noncommittal about specific media or organizations of the interaction. They were more concerned with issues of conceptualization rather than how these should be resolved at the level of input and output implementation. For example, the DEs indicated that they thought graphics I/O methods would be of real value only when the user already had a spatial-visual framework for conceiving the problem domain. There was only one consistent media-related constraint mentioned: that minimal typing should be required to achieve desired functionality.

The higher-order issues the DEs commented on were:

a. Responsiveness. Poor responsiveness may inhibit effectiveness for at least some users. It was not clear from the comments whether the main problem would be in impeding the user's desired rate of interaction or in confusing or irritating him by slow or variable feedback to his inputs.

b. Flexibility. Alternative or rich modes of input are unnecessary because Navy personnel are accustomed to schematic fixed artificial languages. Flexibility in handling errors is desirable.

c. Context. Preservation and display of prior interactions is useful. Maintenance of context across errors and user exercise of initiative is critical. There were no comments regarding the degree of context preservation desirable between work sessions.

d. User Control. The two groups of DEs had somewhat divergent views about user control. The difference may reflect whether the respondents were focusing on introductory or repeated use of a system. The NPRDC DEs favored considerable system control of interactions. They seemed to be considering initial system accessibility and conceptual support for a user. The FCTC DEs favored global user control of interactions with system control exercised locally within particular types of interactions. They seemed to be considering long-term use of systems and their opinions may have been influenced by experience with systems having poor implementations of system control. Both groups believed that user control to escape a system-driven interaction is necessary. Much of their concern with context management centered on supporting this type of user control.

e. User Knowledge Requirements. The DEs thought that a knowledgeable user ought to be able to use a new system effectively in approximately one hour. To do so, they believed that off-line documentation must be organized effectively and that on-line help should always be available. Their strongest comments regarding user knowledge requirements concerned transparency. They believed that transparency is instrumental to reducing the amount of new knowledge required to access a system. We see this as a further statement about the need for conceptual support if a system is to be used effectively.

Most of the comments collected on the feasibility of an IKAS came from the system builders. They all believed that a generic IKAS for knowledge-based systems is not currently possible nor may it ever be. Their beliefs, like ours, derived from the perceived dependencies among system objectives and capabilities, required knowledge, representation formalisms, and knowledge elicitation methods. Some system builders thought that IKAS development with more limited applicability was, however, a worthwhile goal. Such research could aid future system building and maintenance efforts. More fundamentally, it could contribute new knowledge to the field of knowledge-based systems technology.

Comments from most system builders and DEs supported the concept of a class-generic IKAS as a feasible focus for research on techniques to support knowledge acquisition. They agreed that the concept of classes of tasks was at least intuitively viable. The system builders believed that there are difficult problems to be solved if the IKAS system is to use class-generic knowledge to support its interactions, but that solutions to these problems are possible given the current state-of-the-art.

The greatest concern about the IKAS concept involved the question of whether DEs have and can articulate the types of knowledge an IKAS would be designed to obtain. There was consensus among the system builders and DEs that, at least in the specialties with which they are familiar, DEs do not have good causal, diagnostic knowledge associating performance errors with knowledge deficiencies. The system builders also questioned DE abilities for

formulating instructional interventions sensitive to the individual needs of trainees. The DEs were confident, however, that they had been [illegible] knowledge about some errors that might be observed during performance and explanations for those errors in terms of problem features.

RECOMMENDATIONS FOR IKAS DESIGN. As we have stated, the ability to specify the design of an IKAS, particularly its user interface, depends on the capabilities and architecture of the host knowledge-based instructional system to be served by the IKAS. The recommendations we can offer here, based on our interactions with DEs and system builders, are thus limited to [illegible] that should be considered when the other requirements and constraints for developing a particular design are obtained.

a. Emphasize conceptual support and transparency. The IKAS should help the DE-user recognize or learn about its relationship to the host instructional system, its model of the user's task in interacting with it, and its intended model for describing the knowledge about the user's area of expertise. Its interactions should conform to the level of abstraction defined by these frameworks and models and hide lower-level implementation details from the user. A particularly important form of conceptual support is to enable the user to anticipate how knowledge he specifies to the IKAS will manifest itself in the host system's behavior.

b. Design the global knowledge acquisition strategy to be modular with respect to achieving acquisition of different types of knowledge. For some specialties or for an DE within a given specialty, certain types of knowledge may not be known to the user (e.g., rules for causal diagnosis of performance errors). The system should be able to obtain other types of knowledge from the DE and leave the definition of knowledge the DE cannot articulate to other methods.

c. Provide on-line help/documentation facilities and intelligent context management. These interface facilities seem most important for insuring accessibility and effective use by relatively computer-sophisticated DEs, such as those interviewed.

d. Defer commitments on other user interface issues to an incremental design and development approach. Although some of the DEs indicated a preference and need for a fully user-friendly interface, they did not offer an integrated picture of what that would be. Since they have demonstrated their ability to use complex interfaces, they should be able to use most systems as long as they deliver some conceptual support. We believe that the best approach to developing the interface design is through managed refinement of a flexible prototype, based on interaction with DEs using that prototype. This approach allows an empirically based resolution of the tradeoffs between features and system responsiveness. The approach ultimately requires prototype development using hardware and software with potential multi-media capabilities to support alternative input modes and modes of organization for interactions--for example, high-performance, single-user work stations with high-resolution graphics and multiple input media. However, initial work on conceptual support and on higher-level interface

functionality could proceed with less extensive hardware-software resources using a more modest and limited physical interface.

SECTION VI

THE IKAS ARCHITECTURE

This section presents a high-level architecture for implementing the IKAS concept introduced in Section IV. The architecture was designed to support the most useful features developed in prior efforts to support knowledge acquisition (see Section III) and to satisfy the requirements derived from our discussions with Navy personnel (see Section V).

The structure and level of detail of the architecture is compatible with all three alternatives for IKAS development described in Section IV. In fact, it could serve as a framework for any stable, class-generic knowledge acquisition system, instructional or otherwise. Any more detailed specification of the architecture would require commitments to specific representation formalisms, user interface mechanisms, and other mechanisms dependent on the domain characteristics and capabilities of the host knowledge-based system it would serve.

Figure 2 shows the major functional modules of the design and general descriptions of the information they exchange. We first discuss each module's function in the architecture and then discuss the design constraints the architecture aims to satisfy. For each of the three user-accessible modules (the ELICITOR, the CRITIQUER, and the EXERCISER), we provide examples of the functions for the three IKAS concepts described in Section IV.

IKAS MODULES

ELICITOR. The ELICITOR generates suggestions and requests for types of knowledge to be elicited from the user when the system is controlling the interaction. It does so by operationalizing an agenda created by the PLANNER, converting the agenda specifications into requests to be sent to the user across the user interface, and then delivering these requests either by its own mechanisms or those of the USER ASSISTANT. Agenda specifications would include several types of activities. The ELICITOR might suggest a focus of attention on some region or dimension of the knowledge base. Alternatively, it might request particular inputs required for (a) structural completeness or (b) consistency with the semantic and pragmatic features of the domain and of the prior context. The ELICITOR might also suggest appropriate times for the user to invoke the EXERCISER to test the effects of modifications to the knowledge base.

The ELICITOR should be able to explain its behavior to the user using a rationale created along with the agenda. The user can ask why the ELICITOR is following the current tack, and, on the basis of the response, decide whether he wants to comply or take the initiative himself. For instance, the ELICITOR may be able to tell the user it is asking for types of knowledge in a particular order because that order proved effective in eliciting knowledge in other domains.

Examples.

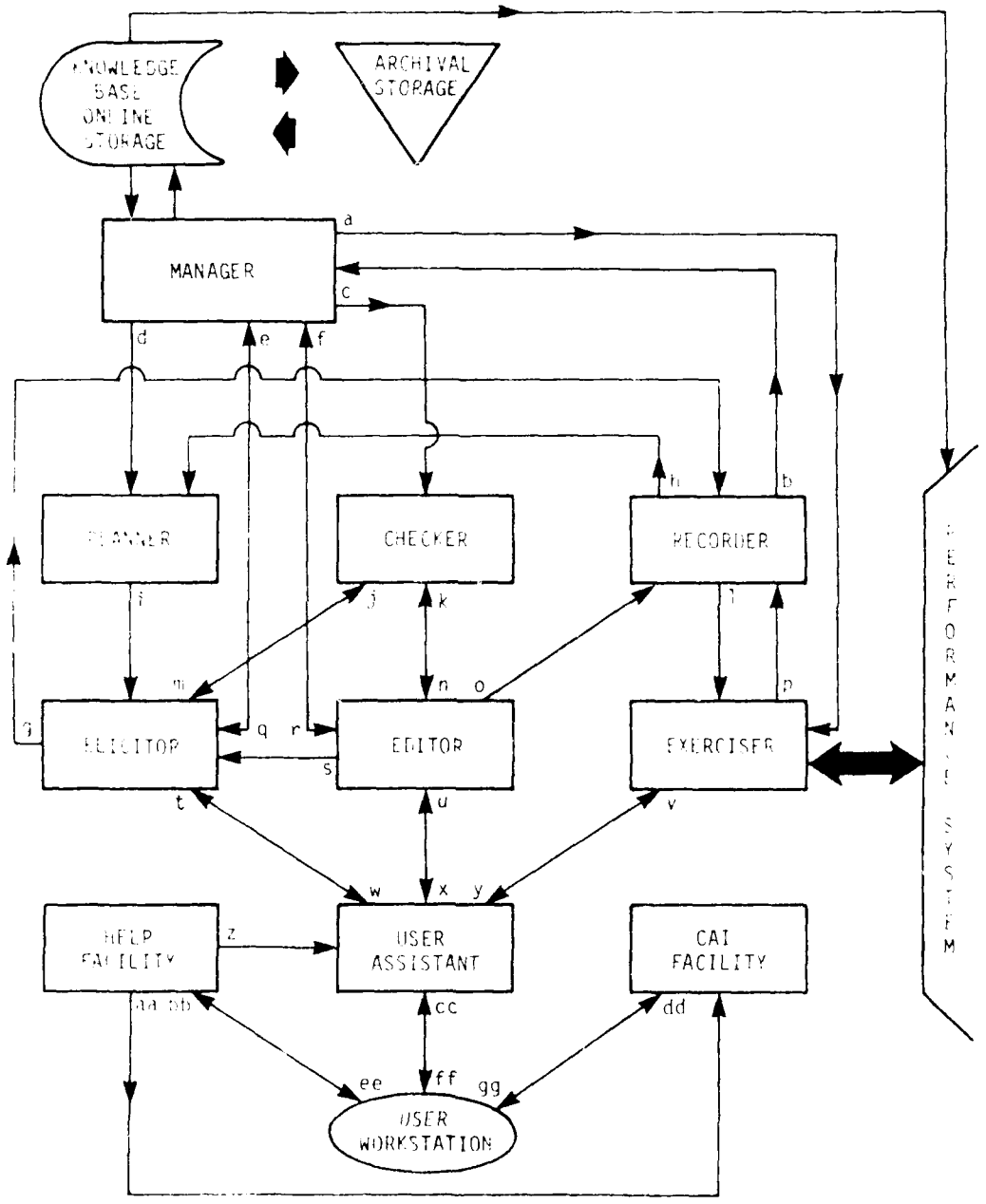


Figure 2. IKAS architecture

a MANAGER -> EXERCISER
Pointers to information in the knowledge base

b RECORDER -> MANAGER
Information pertinent to knowledge base checkpointing and archiving decisions

c MANAGER -> CHECKER
Pointers to information in the knowledge base

d MANAGER -> PLANNER
Pointers to information in the knowledge base

e MANAGER -> ELICITOR
Pointers to information in the knowledge base

f MANAGER -> EDITOR
Pointers to information in the knowledge base

g ELICITOR -> RECORDER
User-System interaction record

h RECORDER -> PLANNER
Records of recent interaction context

i PLANNER -> ELICITOR
Elicitation agenda and rationale

j CHECKER -> ELICITOR
Results of knowledge check

k CHECKER -> EDITOR
Results of knowledge check

l RECORDER -> EXERCISER
Prior exercise definitions and results

m ELICITOR -> CHECKER
Knowledge-base modification to be checked

n EDITOR -> CHECKER
Knowledge-base modification to be checked

o EDITOR -> RECORDER
Interaction records

p EXERCISER -> RECORDER
Exercise records

q ELICITOR -> MANAGER
Request for knowledge base pointer
Changes to current knowledge base

r EDITOR -> MANAGER
Request for knowledge base pointer
Changes to current knowledge base

Figure 2 (cont.). IKAS architecture

s EDITOR -> ELICITOR
User response to ELICITOR query achieved via EDITOR use

t ELICITOR -> USER ASSISTANT
Queries to user
Error information

u EDITOR -> USER ASSISTANT
State information
Error information

v EXERCISER -> USER ASSISTANT
State information
Exercise results for user
Error information

w USER ASSISTANT -> ELICITOR
User responses to information requests
User requests to restore prior state or
modify prior inputs

x USER ASSISTANT -> EDITOR
User commands
User requests to restore prior state or
modify prior inputs

y USER ASSISTANT -> EXERCISER
User requests to test system performance

z HELP FACILITY -> USER ASSISTANT
Pointers to on-line documentation

aa HELP FACILITY -> CAI FACILITY
Pointers to on-line documentation

bb HELP FACILITY -> USER WORK STATION
Documentation requested by user

cc USER ASSISTANT -> USER WORK STATION
Information passed by ELICITOR, EDITOR, EXERCISER
Responses to requests and commands trapped by
USER ASSISTANT

dd CAI FACILITY -> USER WORK STATION
Instructional content

ee USER WORK STATION -> HELP FACILITY
Requests to examine system documentation

ff USER WORK STATION -> USER ASSSITANT
Inputs to be passed to ELICITOR, EDITOR, EXERCISER
Commands and responses to USER ASSISTANT

gg USER WORK STATION -> CAI FACILITY
Inputs to instructional interactions

Figure 2 (cont.). IKAS architecture

a. Alternative 1. The ELICITOR can determine or suggest whether the user should focus on (1) sequentially describing performance characteristics, situation characteristics, or mappings of observables to deviations from the competence model, or (2) all of these for successive segments of the competence model. It can ask about situation characteristics when particular performance characteristics have been specified. It can suggest that the user invoke the EXERCISER when an unfamiliar form of variation from the competence model has been specified as an explanation of a set of performance characteristics.

b. Alternative 2. The ELICITOR can determine or suggest whether elicitation should be organized according to situations or to procedures and rules in the existing opponent simulation. When focusing on a particular situation, it can ask about whether specific rules and procedures, applied in "similar" situations, might also be applied in that situation.

c. Alternative 3. The ELICITOR can determine or suggest whether elicitation should pursue elaboration of taxonomies or the attributes of the concepts in the taxonomies. It can ask whether features associated with a concept are also associated with concepts that are "close" in the taxonomic structure. It can suggest that the user invoke the EXERCISER when a concept with new types of attributes is described.

EDITOR. The EDITOR allows the user to build, modify, and inspect the knowledge base under his own initiative. It is also available through the ELICITOR as a mode for responding to some requests for selection of elements in the knowledge base and for description of modifications to knowledge base structure.

The EDITOR supports transparent, structure-oriented specification of attention and modification. That is, the granularity of its commands is consistent with the syntax and semantics of the knowledge-base formalisms, not the low-level software- and hardware-dependent implementations of those formalisms. The KAS (Reboh, 1981) network editor is a good model for such functionality. Additional support for handling syntactically invalid commands supplied via open input mode is obtained through the EDITOR's interface with the USER ASSISTANT, while support for handling semantic and pragmatic errors is obtained from the CHECKER. Error checking is performed on each input to provide the user immediate feedback.

The EDITOR also must provide context display of the local focus of attention within the global knowledge base. If the representation formalisms for a domain class are conducive to graphic display and manipulation, context display might comprise dynamic "maps" of the knowledge base displayed concurrently with the local editing window. In addition to context display, the EDITOR must also enable the user to locate existing knowledge and change attention by specifying partial descriptions. Other context management functions, such as maintenance and display of alternative contexts, are handled by mechanisms external to the EDITOR.

Examples.

a. Alternative 1. A schema-based editor is possible for descriptions of performance and situation characteristics embedded within a network editor if those descriptions are hierarchical. The frames and schemas are manipulable as slots and values of defined types. The syntax of each defined type determines the granularity of access to values of that type. A network, procedure, or rule-based editor would be used for elaborating deviations of the competence model, depending on how it is represented.

b. Alternative 2. An editor allowing manipulation of networks, procedures, and rules would be used for elaborating alternatives to the opponent simulation model. A network editor would allow manipulations on nodes (procedure or rule designators) and links (control paths) to specify deletion, insertion, and reordering of node invocation. A procedure editor could be oriented toward modification of defined procedures rather than composition from scratch. A rule-editor would allow the user to manipulate the specification and logical composition of primitive clauses in the condition and action portions of the rules governing opponent behavior. The syntax of the clauses would be used to determine the granularity of access to their components (e.g., as members of tuples in predicate-object-value conditions).

c. Alternative 3. A network editor would be used to specify concept and feature taxonomies. It could be combined with a schema-based editor to permit specification of concept attributes and attribute values. A graphics interface could be used to support the network editor and would be advantageous for its context display capabilities.

EXERCISER. The EXERCISER gives the user access to the host KBIS. Its major function is to provide conceptual support through feedback about the relationship between user modifications of the knowledge base and behavior of the KBIS.

A variety of EXERCISER capabilities might be implemented. Most simply, the IKAS user may be allowed to access the KBIS through its instructor or student interfaces to assess system behavior as he changes the knowledge base. Additional control over configuring the state of the KBIS would allow the user to directly configure and test an "interesting" situation he wants to examine. As in some expert consultation systems, the user could draw from a library of problem cases or use an editor to alter these cases in order to exercise the KBIS and obtain summaries of results. In addition, the EXERCISER could be used to automatically select entries from the library according to a set of heuristics. These heuristics would seek to thoroughly exercise new or modified knowledge entered by the user. The requirements for and feasibility and implementation of these EXERCISER capabilities are a function of the host system's capabilities and implementation.

Examples.

a. Alternative 1. The EXERCISER could be used to determine the range of behaviors classified with a given set of performance characteristics. After postulating certain errors and circumstances in

which they should occur, the user could observe performance of the model on problems for which the anticipated errors should occur. He could also test the reliability of performance diagnoses as the knowledge base grows.

b. Alternative 2. The user could observe performance of variants of the opponent simulation in different tactical situations. He could also observe the performance of all the variants in a particular situation. Thus,the EXERCISER could be used to determine the range of behaviors the opponent could exhibit and the conditions under which different behaviors would be invoked by the KBIS.

c. Alternative 3. The user could observe use of his defined concepts in different instructional games. He could observe the games' sequential behavior for a set of concepts. In each case, he could examine the selection rules used by the games in accessing the taxonomies and concept definitions.

RECORDER. The RECORDER acts as a common store of recent context about activities in the ELICITOR, EDITOR, and EXERCISER. This information supports the PLANNER in its maintenance of an agenda for interaction topics. It provides that same context to the EXERCISER to support any bookkeeping or other EXERCISER functions that depend on knowledge of interaction history. The nature of the stored "context" is dependent on the functions of the ELICITOR, EDITOR, and EXERCISER. These will vary from application to application. Generally, "context" refers to the focus of attention within the knowledge base, what effects were achieved at that focus, and how they were achieved.

The RECORDER also provides results of EXERCISER invocations to the MANAGER. These results are used by the MANAGER to make decisions about archiving of the knowledge base in physical storage.

PLANNER. The PLANNER generates the agenda used by the ELICITOR. It uses heuristic rules that operate on the knowledge base and the information maintained by the RECORDER to produce a set of proposed topics and user queries to stimulate the growth of the knowledge base. The PLANNER's rules embody knowledge (acquired by the KEs who engineer the first KBIS for the domain class) about the syntactic structure and class-generic semantic and pragmatic knowledge of the domain. This knowledge can be used to define dialogue themes and transitions that are sensible in terms of their conceptual relationships and importance. They also embody knowledge about more general aspects of dialogue management--for example, the need to vary content and style to avoid user boredom.

Since the IKAS design is oriented toward a mixed-initiative approach to control, the PLANNER needs to plan an agenda only for a short time horizon. As items from the agenda are executed, the PLANNER is invoked whenever the user has problems fulfilling the ELICITOR's requests or assumes the initiative by invoking the EDITOR. At that point the PLANNER replans the agenda. If the user should complete the local agenda, then the PLANNER is invoked to plan for another short time horizon. This dynamic planning and replanning for a short time horizon insures that the ELICITOR's behavior is always sensitive to

recent events. In particular the ELICITOR should follow up on episodes of user initiative.

The PLANNER saves a rationale for the agenda. The rationale describes for each agenda element which rule or rules caused it to be included and what features of the knowledge base or context caused those rules to be applicable. This is similar to the rationale provided by rule-based expert consultation systems. The rationale can be used by the ELICITOR to justify its behavior in response to user request. The ability to supply rationales should make the user more confident in the ELICITOR's behavior and provide information the user can weigh in deciding whether or not to invoke the EDITOR to work under his own initiative.

CHECKER. The CHECKER provides error checking for knowledge base modifications entered through the ELICITOR and the EDITOR. It detects and reports to those modules structural and semantic anomalies in the specifications. Such anomalies may include a modification that would make a taxonomy circular, a value for an attribute that is logically inconsistent with the value for a related attribute, or a rule that will never execute because its activation conditions are subsumed by other rules. The use of closed input modes by the ELICITOR may preclude some such errors from occurring, and the ELICITOR and EDITOR may perform some error checking on their own. However, such checking will be local to the focus of attention. Generally speaking, the CHECKER's role is to provide more global checking against the existing contents of the knowledge base to prevent modifications that would produce errors when considering the overall definition of the knowledge base. This capability requires that the CHECKER incorporate meta-knowledge about the structures and syntax of the knowledge base. Such error checking has been a feature of the knowledge acquisition support of several expert consultation systems (see Section III).

In order to avoid the compounding of errorful specifications, the CHECKER operates on the contents of each modification entered via the ELICITOR and EDITOR. Following such checks, the EDITOR and ELICITOR forward modifications to the MANAGER. Information about CHECKER rejections is forwarded to the RECORDER since it may be of use in planning the elicitation agenda.

MANAGER. The MANAGER frees the user and the other modules from the responsibility of coordinating the management of the knowledge base and its physical storage. Rather than operate on separate, local copies of the knowledge base, the modules share a single virtual knowledge base maintained by the MANAGER. This centralized management appears efficient for supporting mixed-initiative elicitation where both the ELICITOR and the EDITOR are used in an interleaved manner.

The MANAGER must maintain a chronology of knowledge bases so that the ELICITOR or the user (through the EDITOR or EXERCISER) can access and perhaps restore prior contexts. The objective is to allow the user to examine and use prior or alternative knowledge bases and to protect the user from loss due to system errors. The physical storage state of earlier knowledge bases and the method for representing successive variations is invisible to the modules and user and depends on knowledge the MANAGER has about the underlying operating system and hardware. The cost of the capability to restore any arbitrary

prior state would appear too high. In the short term, the RECORDER and USER ASSISTANT may contain information sufficient to restore to any recent context, but in the longer term some intermediate state information needs to be discarded. The MANAGER preserves snapshots of the knowledge base at points specified by the user (through the EDITOR), the ELICITOR, and the RECORDER. Those modules include heuristic rules for determining critical junctures at which a complete long-term record of context may be required. The MANAGER may also determine requirements for saving a full context based on its information about system state.

USER ASSISTANT. The USER ASSISTANT monitors and supports all of the user's interactions with the ELICITOR, EDITOR, and EXERCISER. It is modeled roughly on the capabilities for low-level user support provided in the INTERLISP programming system (Teitelman, et al, 1978). These capabilities include correction of spelling and other simple syntax errors; examining, redoing, and undoing recent events; support for user-defined procedures and abbreviations ("macros"); and ready access to context-sensitive help documentation. These capabilities reduce the consequences of errors and allow the user to operate more efficiently by focusing on his conceptual task rather than low-level communication with the system. Instead of supporting each of these capabilities within the user-accessible modules, they are achieved uniformly within the USER ASSISTANT in order to promote greater consistency.

The USER ASSISTANT operates by trapping all I/O with the user, examining it to determine whether any of its procedures are applicable, recording it, and passing it on. Its capabilities depend on its knowledge about the subsystems with which the user interacts (e.g., spelling correction lists, syntax for open input modes, inverse operations for undoing prior events, and pointers to information in the help facility).

HELP FACILITY. The HELP FACILITY is a documentation database for the IKAS used by the user, the CAI FACILITY, and the USER ASSISTANT. It supports two types of interaction. One, available only to direct use by the user, provides a documentation "tree" through which the user can browse in a more or less top-down manner. In this mode, the documentation provides a well-organized on-line reference manual. The second type of interaction is query-based and is available both to the user and the other modules. Inputs in a query language are used to specify database searches and the result is given to the invoking source.

Both the query language and the result are structured in a machine-readable form to allow uniform use by the other modules. Those modules determine how to display or otherwise use the results in their own interaction context. The HELP FACILITY has its own user interface for allowing the user to compose queries in a more natural mode of expression or for displaying results in a consistent human-readable form. The input interface includes both closed (menu) and open (command language) input mode alternatives for flexibility.

CAI FACILITY. The CAI FACILITY uses an ad hoc frame-oriented approach (the CAI analog to a programmed text) for delivering a tutorial introduction on the use of the IKAS. Much of the content of instruction is retrieved from the

documentation database in the HELP FACILITY. Additional content, such as examples and questions to test understanding, are stored as a separate curriculum data base within the CAI FACILITY.

In our design, the CAI FACILITY is not integrated with the main IKAS modules. Thus, it can not dynamically invoke those modules to present "live" examples, nor can it obtain any information about the errors the user makes in initially using the system, when he may switch between "playing" with the system and accessing the CAI FACILITY. It is limited therefore to simulated examples and in its responsiveness to the user's particular situation.

We do not believe that a more sophisticated CAI FACILITY is necessary for those IKAS concepts that entail significant interaction between the DE and KE prior to DE use of the IKAS (i.e., Alternatives 1 and 2). Under Alternative 3, a more integrated, powerful CAI module may be required to provide self-contained initial access by users. Work on an initial IKAS development effort can, however, proceed independently of requirements for CAI capabilities.

SYSTEM FEATURES

The system architecture described above is intended to satisfy several design constraints. These include:

MIXED INITIATIVE. Knowledge acquisition is either system-directed via the ELICITOR or user-directed via the EDITOR. The user can take control of initiative at any time or pass the initiative to the ELICITOR.

DYNAMIC CONTROL OF INITIATIVE. The flexible mixed-initiative interaction is made possible by dynamic planning of the system's knowledge acquisition objectives. The PLANNER uses context information saved by the RECORDER to maintain an agenda of knowledge acquisition topics. Knowledge acquisition objectives can thereby be altered as necessary to reflect the outcome of prior interactions, whether they were system- or user-controlled.

CONCEPTUAL SUPPORT FOR THE USER. The ELICITOR uses class-generic knowledge from the knowledge base and domain-specific knowledge already entered by the user to generate and interpret user inputs using abstractions consistent with the user's description of the domain. The EXERCISER enables the user to inspect and invoke the host performance system--the KBIS--to test the performance of the system using the current knowledge base. In addition, the HELP and CAI facilities can include reference and tutorial information for supporting the user's understanding of the system.

MODULARITY. Functions for changing and using the knowledge base share common resources for supporting user access, for checking inputs for possible errors, for recording context, and for actual access to the permanent knowledge base. Different aspects of these functions are accomplished by more than one resource (e.g., context management as a function of time by the RECORDER and the MANAGER). Modularity is of course desirable in almost all system architectures. One advantage of modularity in this application is that it should enable incremental system development, thereby enhancing feasibility.

It also provides both greater consistency and flexibility in the user interface.

CONSISTENCY. The MANAGER mediates all interactions between the various modules and the knowledge base. Both the MANAGER and RECORDER free the user from concerns about how to access and update the knowledge base by providing a uniform access mechanism to all the modules. Differences in implementation of knowledge base access from these modules should thus be invisible to the user. For error checking and context recording, the CHECKER and RECORDER should apply uniform mechanisms regardless of whether they are invoked by the ELICITOR (system initiative) or the EDITOR (user initiative). The ELICITOR where possible allows user inputs to be accepted via the EDITOR, providing the user with a consistent input interface under both system and user initiative. Finally, the USER ASSISTANT provides consistent, low-level monitoring and intervention for all user inputs to the major user-accessible functions.

FLEXIBILITY. Flexibility is inherent in the user's ability to shift the responsibility for initiative at any time to use either a special ELICITOR input protocol or the EDITOR when the ELICITOR is in control. Additional flexibility can be achieved through the USER ASSISTANT, which can implement low-level lexical and syntactic error correction and support a consistent set of alternative I/O protocols (e.g., using multiple media, partial input specification, display formats) for the user to select when interacting with the three major user functions.

TURN-KEY ACCESSIBILITY AND USE. Suitable implementation of the USER ASSISTANT, the HELP FACILITY, and the CAI FACILITY could make the system's use independent of external documentation and human support. The major burden is on the CAI facility to help the user understand the performance system's representation of the domain if that understanding cannot be developed via prior interaction with a KE.

SECTION VII

CONCLUSIONS

During the course of this project, considerable effort addressed a careful analysis of the role of the knowledge engineer (KE) in the process of building an expert system. This analysis was directed toward the determination of feasible concepts for automating functions now performed by human KEs. The nature of the knowledge engineering process and the practitioners' current understanding of it appear to limit, at least for the foreseeable future, the scope and generality of possible automation of the KE's role in knowledge acquisition.

The KE uses knowledge and information that is incomplete, inconsistent, and heuristic in attacking his objectives. Part of the problem lies in the communication gap that exists initially between the KE, the customer, and the DE. Another part is due to the fact the knowledge-based systems technology--although it has been applied to some real-world problems--is still immature, lacking the breadth and depth of applicability needed for the emergence of systematic, general methods. As a result, the knowledge engineering process is iterative and incremental, with experience gained early in the process used in subsequent stages to refine and revise system objectives and behavior.

The iterative, incremental nature of knowledge engineering implies that knowledge acquisition, a single objective in the entire process, cannot be isolated from other objectives by automation without interfering with the KE's pursuit of those other objectives. Thus, stand-alone or near stand-alone automation for knowledge acquisition in building a knowledge-based system is not feasible at the present.

Given this conclusion, we developed a system concept for automating knowledge acquisition that avoids a general, comprehensive approach in favor of a more feasible, usable alternative. The concept is based on the notion of a class of related tasks. It proposes that once a knowledge-based system is implemented "manually" for one task in the class--thereby achieving all the KE's objectives--it would be possible to automate elicitation from domain experts of knowledge bases for other tasks in the class. These knowledge bases would be used in a system with the same capabilities and architecture as the first system. The knowledge acquisition mechanisms would make use of knowledge and information obtained in the first effort; hence, they would be specific to that knowledge-based systems architecture for that class of tasks. Automated knowledge acquisition would still proceed incrementally and iteratively throughout stages of formalization, implementation, and testing. However, only the knowledge base of the system would be affected by this process: all other aspects of system design and function would remain fixed.

Although limited in scope and generality, this approach to automating knowledge acquisition should be worthwhile for classes with many members or for systems where new knowledge must be added frequently over the life of the system. It builds upon prior research on assistance for knowledge engineering, which adds to its credibility. However, it involves solving significant new problems. Perhaps the most important of these are (1) how to identify, formalize, and use class-generic abstractions, and (2) how to

provide conceptual support to users that would enable DEs to use the system directly.

The three alternative applications of the automated knowledge acquisition concept presented in Section IV have different costs and expected benefits according to the criteria considered. However, each appears to present a promising and productive avenue for realizing new capabilities in automated knowledge acquisition. Non-technical considerations regarding organizational needs or synergy with on-going research programs may ultimately influence which alternative would be most fruitful to pursue. Although our concept definitions were shaped primarily by technical considerations, we also were influenced by our perceptions of those contributions that would provide high value to the Navy at this time.

We also determined that a detailed design of the user interface for an IKAS is not possible without commitment to a specific, detailed KBIS-IKAS architecture. However, our discussions with Navy system builders and DEs indicated that, while the details of an IKAS' user interface are not inconsequential, they will probably not determine whether Navy DEs can effectively use initial IKAS technology. Instead, selection of users and high-level user interface characteristics not identified with particular media or interaction protocols are more critical for a successful IKAS implementation for a Navy training system. We therefore made the following recommendations, reflected in the architecture proposed in Section VI, for pursuing further design of an IKAS:

a. Emphasize conceptual support and transparency.

b. Design the knowledge acquisition strategy to be modular with respect to acquiring different types of knowledge.

c. Provide extensive on-line help/documentation facilities and intelligent context management.

d. Defer commitments on other user interface issues in an incremental design and development approach.

We believe that by following these recommendations further research can build upon the results of the present project to develop a first effective prototype IKAS.

REFERENCES

Alperovitch, Y. A knowledge acquisition system for plan structuring. Doctoral dissertation, University of California, Los Angeles, 1982.

Barr, A., and Feigenbaum, E. (Eds.). The handbook of artificial intelligence (Vol. 2). Los Altos, CA: William Kaufmann, Inc., 1982.

Bennett, J. S. Personal communication.

Brown, J. S. Uses of artificial intelligence and advanced computer technology in education. In R. J. Seidel and M. Rubin (Eds.), Computers and communications: Implications for education. New York: Academic Press, 1977.

Buchanan, B. G. Research on expert systems (Report No. STAN-CS-81-837). Stanford, CA: Department of Computer Science, Stanford University, 1981.

Buchanan, B. G., Barstow, D., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. A. Constructing an expert system. In F. Hayes-Roth, D. Waterman, and D. Lenat (Eds.), Building expert systems. Reading, MA: Addison-Wesley, 1983. In press.

Clancey, W. J. Methodology for building an intelligent tutoring system (Report No. STAN-CS-81-894). Stanford, CA: Department of Computer Science, Stanford University, 1981.

Crawford, A. M., and Hollan, J. D. Development of a computer-based tactical training system (Special Report 83-13). San Diego, CA: Navy Personnel R&D Center, 1983.

Davis, R. Interactive transfer of expertise: Acquisition of new inference rules. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence. IJCAI, 1977.

Duda, R., Gaschnig, J., Hart, P. E., Konolige, K., Reboh, R., Barrett, P., and Slocum, J. Development of the PROSPECTOR consultation system for mineral exploration (Final Report, SRI Projects 5821 and 6415). Menlo Park, CA: SRI International, 1978.

Erman, L., Hayes-Roth, F., Lesser, V., and Reddy, R. The Hearsay-II speech understanding system: Integrating knowledge to reduce uncertainty. Computing Surveys, 1980, 12(2), 213-252.

Fain, J., Gorlin, D., Hayes-Roth, F., Rosenschein, S., Sowizral, H., and Waterman, D. The ROSIE Language Reference Manual (Rand Note N-1647-ARPA). Santa Monica, CA: Rand Corporation, 1981.

Fikes, R. E., and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 1971, 2, 184-208.

Hayes-Roth, B. Human planning processes (Report R-2670-ONR). Santa Monica, CA: The Rand Corporation, 1980.

Lindsay, R., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. DENDRAL. New York: McGraw-Hill, 1980.

Martin, J. Design of man-computer dialogues. Englewood Cliffs, NJ: Prentice-Hall, 1976.

McCandless, T. Computer-based tactical memorization system (Technical Note 81-8). San Diego, CA: Navy Personnel R&D Center, 1981.

McDermott, J. R1: an expert in the computer systems domain. In Proceedings of the 1st Annual National Conference on Artificial Intelligence. American Association for Artificial Intelligence, 1980.

Nii, H. P., and Aiello, N. AGE (Attempt to Generalize): A knowledge-based program for building knowledge-based programs . In Proceedings of the 6th International Joint Conference on Artificial Intelligence. IJCAI, 1979.

Nii, P., Feigenbaum, F., Anton, J., and Rockmore, A. Signal-to-symbol transformation: HASP/SIAP case study. AI Magazine, 1982, 3(2), 23-35.

Pearl, J., Leal, A., and Saleh, J. GODDESS: A Goal-Directed Decision Structuring System. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1982, PAMI-4(3), 250-262.

Ramsey, H. R., and Atwood, M. E. Human factors in computer systems: a review of the literature (Tech. Rep. SAI-79-111-DEN). Englewood, CO: Science Applications, Inc., 1979.

Ramsey, H. R., Atwood, M. E., and Kirshbaum, P. J. A critically annotated bibliography of the literature of human factors in computer systems (Tech. Rep. SAI-78-070-DEN). Englewood, CO: Science Applications, Inc., 1978.

Reboh, R. Knowledge engineering techniques and tools in the PROSPECTOR environment (Technical Note 243). Menlo Park, CA: SRI International, 1981.

Rumelhart, D. Toward an interactive model of reading (Tech. Rep. CHIP Technical Report No. 56). San Diego, CA: University of California, San Diego, 1976.

Shortliffe, E. H. Computer-based medical consultations: MYCIN. New York: American Elsevier, 1976.

Simpson, H. A human-factors style guide for program design. BYTE, 1982, 7(4), 108-132.

Stefik, M., Aikins, J., Balzer, R., Benoit, J., Birnbaum, L., Hayes-Roth, F., Sacerdoti, E. The architecture of expert systems. In F. Hayes-Roth, D. Waterman, and D. Lenat (Eds.), Building expert systems. Reading MA: Addison-Wesley, 1983. In press.

AD-A139 019 ALTERNATIVE KNOWLEDGE ACQUISITION INTERFACE STRUCTURES (U) PERCEPTRONICS INC MENLO PARK CA KNOWLEDGE SYSTEMS BRANCH K T WESCOURT ET AL. DEC 83 PPAFTR-1131-83-1

UNCLASSIFIED NAVTRAEQUIPC-82-C-0151-1 F/G 5/9 NL

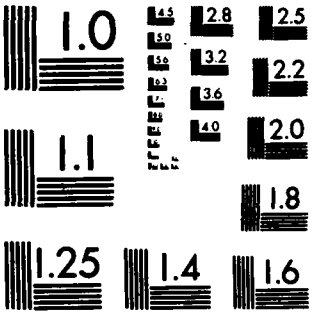
2/2

END

DATE FILMED

4 84

DTIC



Suwa, M., Scott, A. C., and Shortliffe, E. An approach to verifying completeness and consistency in a rule-based expert system. AI Magazine, 1982, 3(4), 16-21.

Teitelman, W., et al. Interlisp reference manual. Bolt Beranek and Newman, Inc. and XEROX Palo Alto Research Center, 1978.

Thorndyke, P. A rule-based approach to cognitive modeling of real-time decision making. In Proceedings of the Workshop on Cognitive Modeling of Nuclear Plant Control Room Operators. Oak Ridge National Laboratories, 1982.

Thorndyke, P., McArthur, D., and Cammarata, S. AUTOPILOT: A distributed planner for air fleet control. In Proceedings of the Seventh International Conference on Artificial Intelligence. IJCAI, 1981.

van Melle, W. A domain-independent production-rule system for consultation programs. In Proceedings of the 6th International Joint Conference on Artificial Intelligence . IJCAI, 1979.

Weiss, S. M. and Kulikowski, C. A. EXPERT: A system for developing consultation models. In Proceedings of the 6th International Joint Conference on Artificial Intelligence. IJCAI, 1979.

Wipke, W. T., Braun, H., Smith, G., Choplin, F., and Sieber, W. SECS--Simulation and Evaluation of Chemical Synthesis: Strategy and planning. In W. T. Wipke and W. J. House (Eds.), Computer-assisted organic synthesis. Washington, D.C.: American Chemical Society, 1977.

APPENDIX A

ILLUSTRATIVE INSTRUCTIONAL KNOWLEDGE ACQUISITION DIALOGUE

This Appendix presents a hypothetical dialogue between a DE and an KE who are working together to define knowledge to be used by a KBIS for surface Navy tactics training. It is intended to illustrate some of the requirements for eliciting knowledge used for trainee performance modeling (see IKAS Alternative 1 in Section IV). The dialogue assumes the KE and DE have been interacting for a considerable time and that a competence model is largely completed.

* * * * *

KE: We've considered the procedures and rules for successful defense of a single ship from surface and air threats. Now let's consider how things can be done wrong if a TAO lacks that knowledge. In particular, I want you to think about training exercises you've supervised and the errors you've frequently seen and what causes them. Let's limit ourselves first to "weapons free" situations. You indicated earlier that decisions were contingent on how strongly you believed your opponent had sufficient data to target you. Are there important errors that you've seen there?

DE: Well, yes. It's mostly a case of keeping track of his and your emissions. I guess one common mistake is to forget about emissions you may have produced before you knew he was there, like HF or even UHF communications. I think I told you that if you detect him passively and you have put out some emissions you should start a zig-zag if you are authorized to.

KE: So in that type of situation, not seeing a zig-zig would lead you to believe, as an observer, that a TAO either had forgotten about the emissions or didn't know that a zig-zag was the thing to do then?

DE: Yes, most of the time. Depending on what type of opponent he believes he is facing, he could decide it was very unlikely that the opponent could get targeting information passively from the emissions. In that case, he might not zig-zag. And remember, you can't always zig-zag; it depends on the formation and on whether you happen to be on a ship with certain types of towed sonar deployed.

KE: Right, I can see that here in my notes about manuevering. You gave me those constraints before.

DE: Remember too the deception angle. If you really think he has targeting data and is just waiting to get a better shot, then you might buy some time to get your own systems ready to fire by NOT zig-zagging. So, it's not all that simple to say from not seeing zig-zagging that a TAO has not been thinking about his emissions. It's a matter of judgment. Like I told you, you don't want to be too cute. If the situation is hot and you gave him some passive

data and now you have some from him, you really want to think about going active first to get targeting data.

KE: So, the way you would really evaluate a TAO's performance is whether he let the opponent get off the first shot on a good intercept heading?

DE: Yes, it's pretty imprecise--sometimes you can weigh the odds as best you can and take the actions you know are best and still come out on the short end. I'd need to see a guy apparently miss the fact that he had been targeted several times before I'd conclude that he was not considering his emissions history or acting on it to the best of his ability. Of course, if it came up in a training exercise, I could ask him about it in the debriefing if it seemed to impact the outcome.

KE: Suppose you wanted to figure out whether a TAO trainee using a training simulator like NAVTAG had this problem just by looking at his performance. What type of situations would you set up?

DE: I'd put him on a low-capable ship and give him some INTEL about some high-capable opponents; that way he wouldn't be quick to go on the offensive without first hoping ORANGE would give him some good data that could compensate for the capability imbalance. I'd put no constraints on his manuevering. Then I'd set him up by having him respond to some communication and at some later time pick up some ambiguous passive emission that was ORANGE-originated but insufficient for identification. Then I could look for a zig-zag with more certainty. I'd give him a few exercises in which those were the features before any engagement actually commenced.

KE: What about his possible belief that the enemy hadn't been able to target him, or an attempt at deception?

DE: Well, on the first, those high-capability ships generally have the best ESM systems, so if his INTEL says that's what he might expect, he wouldn't want to ignore them. Also, instead of just a COMM emission I could set the simulator to give him a real emissions error--tell him his ECM equipment or a weapons control radar accidently went on for a few seconds. That's real unlikely, but possible. As to the deception, if I set up the trainer to ignore his deception and hit him with an SSM if he didn't zig-zag, then after a few exercises he'd be zig-zagging if he knew he was supposed to do so or if he wasn't forgetting his emissions history. Only problem with that is discouraging his use of deception when it might be his best chance. I'd address that directly if I were an instructor after I was sure a trainee knew the best non-deception response to the situations.

KE: Let's talk about situations now where BLUE believes he's been targeted with regard to bearing and wants to initiate the engagement. Those are where he is on a high-capable platform or has an important defensive role for some other HVUs.

DE: That's right. The biggest problems are not giving ORANGE better targeting data before you are ready to fire yourself.

KE: Let's talk first about the case where BLUE hasn't seen a surveillance radar emission from ORANGE.

DE: Well one problem I've seen there is BLUE's use of his own surveillance radar. First, you don't want to use it if your EW people tell you the track's source seemed to be at a range outside your destruction zone. If you do, all you are going to do is confirm your identity and maybe give him track info. It may allow you to establish a better track on him but you can't do anything about it.

KE: You told me that in that case you should steam toward the target's bearing, so that the error is in not executing that procedure when the situation warrants it--that is, when you have suggestive evidence that the target is out of range.

DE: Right. Except of course in the case where your SSMs are mounted aft and you might not be able to recover fast enough to shoot if you are in range and he shoots first. That's the tricky one if you are determined to be offensive.

KE: You said "first" problem before: what other problems are there in using surveillance radar at that point?

DE: It's turning it on continuously right away instead of taking a snapshot. That gives him better targeting and all you need is a snapshot to determine how to engage him with the fire control systems. You don't want to monitor with your surveillance radar unless you've already seen his fire control radar light off.

...

NAVTRAEQUIPCEN 82-C-0151-1

DISTRIBUTION LIST

Naval Training Equipment Center 25
N-71
Orlando, FL 32813

Technical Library
Naval Training Equipment Center
Orlando, FL 32813

Defense Technical Information Cen. 12
Cameron Station
Alexandria, VA 22314

Keith Wescourt 50
Senior Computer Scientist
Perceptronics
Knowledge Systems Br.
545 Middlefield Rd., Suite 140
Menlo Park, CA 94025

CDR Daniel F. Hassett
(Code 32-06)
Fleet Combat Training Center, Pacific
San Diego, CA 92147

LT Skip McVay
(Code 32-06)
Fleet Combat Training Center, Pacific
San Diego, CA 92147

LT William Bloomberg
Fleet Combat Training Center, Pacific
San Diego, CA 92147

LT Richard Coleman
Fleet Combat Training Center, Pacific
San Diego, CA 92147

CDR Dave Anderson
Navy Personnel R&D Center
San Diego, CA 92152

SR CHIEF William Smith
Navy Personnel R&D Center
San Diego, CA 92152

Dr. Alice Crawford
Navy Personnel R&D Center
San Diego, CA 92152

Dr. Barbara MacDonald
Navy Personnel R&D Center
San Diego, CA 92152

Dr. Ed Hutchins
Navy Personnel R&D Center
San Diego, CA 92152

Dr. Jim Hollan
Navy Personnel R&D Center
San Diego, CA 92152

Mr. Robert Bechtel
Naval Ocean Systems Center
San Diego, CA 92152

Dr. Marshall Farr
Office of Naval Research
Code 442PT
800 N. Quincy Street
Arlington, VA 22217

Dr. Henry Halff
Office of Naval Research
Code 442PT
800 N. Quincy Street
Arlington, VA 22217

Dr. Martin Tolcott
Office of Naval Research
Code 442
800 N. Quincy Street
Arlington, VA 22217

Dr. William Vaughan
Office of Naval Research
Code 442
800 N. Quincy Street
Arlington, VA 22217

Mr. Alan Meyerowitz
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

Dr. Robert Sasmor
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Harold F. O'Neil, Jr.
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Joseph Psotka
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Ms. Judith Orasino
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Mr. Edgar Johnson
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Mr. Stan Halpin
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

COL Neale Cosby
Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Mr. Joseph Yasutake
AFHRL (LRT)
Lowry AFB, Colorado 80230

Dr. Genevieve Haddad
Life Science Directorate
AFOSR/NL - Bldg 410
Bolling AFB
Washington, DC 20332

Dr. Jude Franklin
Naval Research Laboratory, Code 7510
Washington, DC 20375

Mr. Gordon Powell
Naval Surface Weapons Center
Indian Head, MD 20640

Mr. Jack Wozencraft
Naval Postgraduate School
Monterey, CA 93940

MAJ Jack Thorpe
DARPA
1400 Wilson Blvd
Arlington, VA 22209

Dr. Robert Kahn
DARPA
1400 Wilson Blvd
Arlington, VA 22209

Mr. Ron Ohlander
DARPA
1400 Wilson Blvd
Arlington, VA 22209

Mr. Norton Fowler
Rome Air Development Center
Griffiss AFB, NY 13441

Mr. Al Barnum
Rome Air Development Center
Griffiss AFB, NY 13441

Mr. Yale Smith
Rome Air Development Center
Griffiss AFB, NY 13441

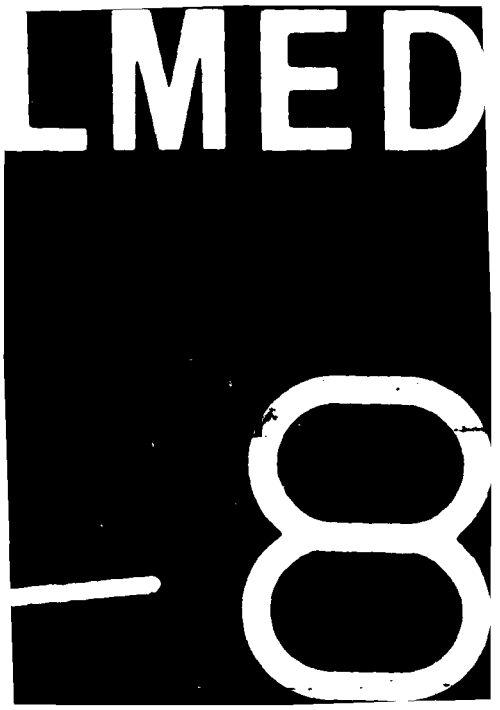
Mr. Richard Metzger
U.S. Army Human Engineering Laboratory
Aberdeen Proving Ground, MD 21005

Dr. Benjamin Cummings
U.S. Army Human Engineering Laboratory
Aberdeen Proving Ground, MD 21005

Dr. John Weiss
U.S. Army Human Engineering Laboratory
Aberdeen Proving Ground, MD 21005

Dr. Henry R. Cook
Defense Mapping Agency
BLDG 56
U.S. Naval Observatory
Washington, DC 20305

Dr. Ronald Hofer
PM TRADE-E
Naval Training Equipment Center
Orlando, FL 32813


LMED
8